Universitat de Lleida

TREBALL FINAL DE GRAU

Estudiant: **Guillem Orellana Trullols**

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: Automatització del cicle de desenvolupament d'un model d'aprenentatge automàtic

Director/a: Ramón Béjar Torres i Francesc Guitart Bravo

Presentació

Mes: Juliol

Any: 2019

Agraïments

En primer lloc, vull agrair els tutors del meu treball, Francesc Guitart i Ramón Béjar, per orientar-me, estar sempre disponible per a resoldre els meus dubtes i per l'interès i recolzament que m'han ofert.

Als meus pares i família per recolzar-me i perquè d'una manera o altra tots han aportat alguna cosa en el treball i en el transcurs de la carrera. Finalment, també vull agrair a la meva parella per donar-me suport i ajudar-me amb l'estructura i la correcció del document.

En definitiva, moltes gràcies a tots els que heu fet possible aquests quatre anys de carrera.

RESUM

Introducció

Python és un dels llenguatges de programació més populars per a la informàtica científica. Gràcies a la seva naturalesa interactiva i a l'ecosistema de biblioteques científiques, és una opció molt atractiva per al desenvolupament algorítmic i l'anàlisi exploratori de dades.

Actualment, un dels mòduls més utilitzats és Scikit-learn, el qual integra un gran nombre d'implementacions actualitzades d'algorismes de ML. Degut a la consistència en la seva API, la comunitat ha estat capaç d'establir fluxos de treball i automatitzar certes activitats basades processos de tractament de dades i entrenament d'algorismes.

Tot i que existeix un elevat nombre d'eines de codi lliure que faciliten el flux de treball d'un problema de ML, no n'hi ha cap a l'abast de la comunitat que faciliti el seguiment de directrius o bones pràctiques i que permeti treballar sota un cicle comú.

Objectiu

Analitzar i estudiar diferents projectes de l'àmbit de ML per extreure'n les parts essencials amb la finalitat d'establir un cicle de desenvolupament genèric per a ser reutilitzat per tota mena de projectes del món de la IA. A més, establir directius i bones pràctiques per tal de garantir resultats estadísticament sòlids.

Proposta tècnica

DriftAI és un mòdul de *Python* publicat com a codi lliure sota la llicència *AppVerse.* El seu objectiu és ajudar els *data scientists* en el seu dia a dia automatitzant el cicle de desenvolupament d'un projecte ML. Algunes de les seves funcionalitats són: estandarditzar l'accés a les dades d'entrenament i validació, permetre iterar entre diferents configuracions de models i optimitzar els hyperparàmetres. A més, DriftAI defineix un cicle de vida comú independentment del problema i de la tecnologia utilitzada per resoldre'l. A més, proporciona una completa reproductibilitat al llarg de les seves fases.

Conclusions

Estandarditzar el cicle de vida de desenvolupament d'un model de ML és complex però alhora necessari. Malgrat les complexitats que involucra aquest procediment, s'ha aconseguit implementar un *framework* que defineix un cicle de vida genèric i extensible.

El cicle de vida definit destaca pels següents punts forts: accés estandarditzat a la font de dades, separació dels conjunts de dades que evita el filtratge d'informació entre el conjunt d'entrenament i validació, extensibilitat per definir nous comportaments, escalabilitat per accelerar l'entrenament i la cerca d'hyperparàmetres interfície comuna per avaluar els diferents models.

Paraules clau

Intel·ligència Artificial, *Deep learning, Machine Learning, Framework*, Desenvolupament Software

ABSTRACT

Introduction

Python is one of the most popular languages among the computer science community. Thanks to its high-level interactive nature and due to the ecosystem of scientific packages which surrounds it, *Python* is a good choice to develop machine learning algorithms and to perform data exploration.

Nowadays, the most widely used package is Scikit-learn. Scikit-learn integrates a large number of state-of-the-art machine learning algorithms. Because of its ease of use and its API's consistency, the community have been able to create and automate a wide range of data preprocessing techniques and training approaches based on Scikit-learn.

Even though a lot of open source tools are already helping developers to create and automate pipelines to solve machine learning problems, there isn't a silver bullet to create new machine learning models with good practices and neither a standardized lifecycle. This involves that every data scientist works with his own reasonable judgement, meaning that no quality is assured.

Goal

Analyze and study different machine learning project and extract the reusable and essential parts in order to define a generic lifecycle which suits to every artificial intelligence project. Furthermore, establish good practices to guarantee successful and statistically verified results.

Technical proposal

DriftAI is an open source Python package published under the *AppVerse* License. DriftAI objective is to help data scientists on their everyday work automating the machine learning models development lifecycle. DriftAI main advantages are: standardized train and validation data access, model exploration with different hyperparameter configurations, hyperparameter tuning, standardized evaluation methods, fully reproducibility, time reduction when create "production" model. Furthermore, DriftAI defines a unique lifecycle regardless the problem or the used technology to solve it.

Discussion

Standardize the whole machine learning project lifecycle is complex but necessary in order to allow your business scale alongside your artificial intelligence services. Despite of the complexities a framework capable of manage a generic and extensible lifecycle have been developed.

The strengths of the defined lifecycle lie on the following features: standardized data source access, no knowledge leaking between train and validation sets, scalability to speed up training and algorithms hyper tune and a common evaluation interface.

Key words

Artificial intelligence, Deep learning, Machine Learning, Framework

ÍNDEX

LLISTA DE TAULES

LLISTA DE FIGURES

LLISTA D EQUACIONS

LLISTA DE FRAGMENTS DE CODI

Acrònim	Significat
AUC	Area Under the Curve
BD	Base de Dades
CD	Continuous Delivery
CI	Continuous Integration
CLI	*Command Line Interface*
CV	*Cross Validation/* Validació creuada
CSV	*Comma Separated Value*
DL	Deep Learning / Aprenentatge profund
FN	False Negatives
FNR	False Negative Rate
FP	False Positives
FPR	False Positive Rate
GANs	Generative Adversarial Networks, Xarxes Neuronals Generatives
GPU	Graphic Processor Unit
IA	Intel·ligència Artificial
MAE	Mean Absolut Error
MAR	Missing at Random
MCAR	Missing Completely at Random
ML	*Machine Learning* / Aprenentatge automàtic
MNAR	Missing not at Random
NN	Neural Network / Xarxa neuronal
OO	Orientada a Objectes
RMSE	Root Mean Square Error

ROC	Reciever Operating Characteristic
PaaS	*Platform as a Service*
Sklearn	Scikit-learn
SVM	Suport Vector Machines
TPU	Tensor Processing Unit
TN	True Negatives
TP	True Positives

1. INTRODUCCIÓ

Durant els últims anys, la Intel·ligència Artificial (IA) ha estat objecte d'un intens sensacionalisme en els mitjans de comunicació. El *Machine Learning* (ML), el *Deep Learning* (DL), i la IA són conceptes mencionats en nombrosos articles, sovint en camps no tecnològics. Actualment la IA promet un futur ple de robots intel·ligents, vehicles autònoms i assistents virtuals, un futur que de vegades és definit com a terrible o d'altres com a una utopia, on les feines que avui en dia realitzem seran pràcticament inexistents i, d'aquesta manera, la majoria d'activitats econòmiques seran dutes a terme per robots o agents intel·ligents.

Tot i que en els darrers anys la IA ha aconseguit assolir objectius notables, les expectatives a curt termini sobre els pròxims assoliments tendeixen a ser massa altes envers el que realment és possible. Encara que existeixin aquestes expectatives idíl·liques i exagerades el futur d'aquest camp a llarg termini és brillant, de fet, l'aplicació de la IA en aspectes de la nostra vida diària només ha fet que començar, ja que actualment ja es treballa amb projectes com els cotxes autònoms, *smartphones* amb IA integrada per a les tasques quotidianes, etc. A més en un futur proper la IA pot ajudar en altres camps com la medicina, facilitant operacions que actualment són complexes, l'art, amb la generació de quadres i/o músiques...

La IA no és un camp d'investigació recent, ja que les primeres publicacions relacionades amb el ML van aparèixer en la dècada del 1950. Per tant, ens podem preguntar per què avui en dia té tanta influència i abans no? L'explicació d'aquest "boom" que ha sorgit en els últims anys és a causa de tres aspectes principals:

- Millora del *hardware*
- Augment de les dades
- Millora en els algorismes

Pel que fa al *hardware,* en els darrers 20 anys les CPUs han esdevingut 5.000 vegades més ràpides (Moore 1975), fet que ha suposat que tothom pugui aplicar tècniques de IA des del seu propi ordinador portàtil. A més, gràcies als avenços de la indústria dels videojocs, les targetes gràfiques dels ordinadors (GPUs) han millorat any rere any de forma exponencial. Aquest avenç va beneficiar els científics quan l'any 2007 la companyia NVIDIA va publicar una interfície de programació per treballar amb les seves GPUs, la qual va ser anomenada CUDA[1]. Treballar amb CUDA permet paral·lelitzar de forma eficient els còmputs d'operacions entre matrius per tal de reduir de forma considerable el temps dedicat a l'entrenament dels algorismes de IA.

[1] https://www.nvidia.es/object/cuda-parallel-computing-es.html

A part de les GPU's, cada vegada s'utilitzen més les *Tensor Processing Units* (TPU)[2]. Les TPU són microprocessadors dissenyats específicament per les aplicacions de la IA, de forma que permeten paral·lelitzar i realitzar operacions amb tensors (matrius de N dimensions) de forma més ràpida i eficient que les GPUs (Jouppi et al. n.d.)

La IA de vegades és definida com la nova revolució industrial i, per tant, si la IA és la màquina de vapor d'aquesta revolució, les dades són el carbó: es tracta de la matèria primària que dóna energia a les màquines intel·ligents sense la qual aquest progrés no seria possible. Així doncs, quan es fa referència a l'augment de dades, es parla que ha estat gràcies a la internet, la qual ha facilitat l'agrupament de dades i formació de nous *data sets* imaginables sense una connexió a escala global.

Addicionalment a les dades i al *hardware*, fa 20 anys no existia una forma fiable d'entrenar eficientment algorismes com les xarxes neuronals, ja que els algorismes d'aprenentatge eren lents i no permetien obtenir resultats estadísticament sòlids. Aquest fet va canviar durant els anys 2009-2010 amb millores importants dels algorismes d'entrenament, les quals van permetre realitzar entrenaments més ràpids i obtenir millors resultats.

Per els motius esmentats anteriorment, la IA ha esdevingut un dels camps més importants en l'àmbit de la recerca, i de forma progressiva va guanyant pes en les nostres vides quotidianes.

De fet la IA està guanyant tanta importància en el món de la informàtica que en Geoffrey Hinton, en Yann LeCun i en Yoshua Bengio, també coneguts com a "*godfathers of artificial intelligence*", recentment han guanyat el premi *Turing*, també conegut com el premi *Nobel* de la informàtica gràcies a la seva constància i avanços obtinguts en la IA.

Així doncs Hinton, Lecun i Bengio han han contribuït en el desenvolupament de investigacions com ara:

- La creació de l'algorisme de la retropropagació, el qual es la base de l'entrenament de les NN actualment (Hinton, Rumelhart, and Williams 1986).
- La creació d'arquitectures de xarxes convolucionals (Lecun et al. 1998).
- La creació de les xarxes neuronals generatives adversaries(GANs).

(Goodfellow et al. 2014)Tot i que en els anys 80 i 90 la va experimentar un creixement de popularitat i comunitat científica, a mitjans del 90 els científics van fracassar en la majoria d'investigacions i es van realitzar pocs avanços, fet que va provocar que es disminuís la inversió

[2] https://cloud.google.com/tpu/

destinada a quest camp. Malgrat aquesta disminució d'esperança en la IA, Hinton, LeCun i Bengio van continuar apostant per ella.

Finalment, l'any 2012 tot i el poc finançament del que disposaven van proposar un algorisme anomenat xarxa neuronal (Neural Network (NN)) que funcionava un 40% millor en tasques de classificació i regressió que la resta d'algorismes que s'estaven utilitzant fins al moment.

Actualment, els cotxes autònoms, els assistents de veu, el reconeixement facial dels *smartphones,* etc. Són tecnologies que són possibles gracies als avanços fets per Hinton, LeCun, i Bengio[3].

[3] https://amturing.acm.org/

1.1 Justificació i interès del tema

Degut a que el camp d'investigació de ML evoluciona molt ràpidament resulta complex establir normes i/o estàndards. Per exemple, en camps de la informàtica que ja disposen de cicles i metodologies ja preestablertes, com ara el desenvolupament de *software,* existeixen un conjunt de bones practiques i tècniques molt sofisticades per comprovar i trobar errors abans de la posta en producció d'un sistema. Malauradament no existeix una practica equivalent per a problemes resolts amb ML, ja que degut a la gran escala i la manca d'estructura que tenen aquests models, els quals poden contenir milions de paràmetres, resulta molt complex poder definir tècniques de *testing* robustes[4]. Es pot contrastar en l'apartat de l'estat del art que no existeix una única ni millor manera d'afrontar un projecte de ML.

Els estàndards són normes que estableixen especificacions i procediments dissenyats per tal de garantir la fiabilitat del producte resultant d'una activitat. Les normes aborden una sèrie de problemes, que inclouen, entre d'altres, diversos protocols que ajuden a garantir la funcionalitat i la compatibilitat dels productes, facilitar la interoperabilitat, etc.

Els estàndards permeten establir protocols coherents que es poden entendre i adoptar universalment, això afavoreix l'enteniment i interoperabilitat entre companyies, o treballadors d'una mateixa empresa. Amb aquest enteniment comú es facilita el desenvolupament del producte, o en el cas del ML, del model predictiu, i s'accelera el temps de comercialització d'aquest. A més, desenvolupar un projecte mitjançant estàndards proporciona més credibilitat al producte resultant[5].

A part dels estàndards internacionals com IEEE[6] o ISO, en l'àmbit de la informàtica existeixen normes que són aconsellables seguir si es desitja obtenir un producte fiable i simple d'entendre per altres desenvolupadors. Per exemple, molts *frameworks* o gestors de dependències estableixen una estructura de directoris comuna per als seus projectes, d'aquesta forma tot desenvolupador pot conèixer l'estructura i adaptar-se ràpidament a un nou projecte.

Si tots els desenvolupadors treballen seguint les mateixes directrius resulta senzill establir un cicle comú entre tots els projectes. Actualment existeixen metodologies que es exploten el potencial que tenen aquests cicles, per exemple, la metodologia *Continuous Integration and Continuous Delivery* (CI/CD) permet automatitzar un cicle basat es basa en les 3 etapes següents:

- *Build*

[4] https://deepmind.com/blog/robust-and-verified-ai/
[5] https://www.iso.org/benefits-of-standards.html
[6] https://standards.ieee.org/

- *Test*
- *Deploy*[7]

Malauradament, en el món de la IA la comunitat no ha estat capaç d'establir un cicle d'aquestes característiques i posteriorment automatitzar-lo. Això es degut a la complexitat que suposa, ja que el cicle d'un projecte de ML comporta realitzar processos molt costosos, com ara tractar una gran quantitat de dades, optimitzar els hyperparàmetres d'un algorisme, etc.

Algunes solucions actuals, com ara *ml-flow*[8] estableixen un cicle basat en les tres etapes següents:

- Entrenament
- Avaluació
- Desplegament del millor model.

Tot i això, com es veu en el marc teòric, un projecte ML es molt més extens i compta amb moltes més fases, les quals resulta imprescindible contemplar en un projecte d'aquestes característiques. A més les solucions més actuals no aporten cap mena de solució tècnica a problemes freqüents que poden aparèixer al llarg de les 3 etapes esmentades anteriorment.

Per tant, el principal interès d'aquest treball és crear un cicle de vida i un conjunt d'estàndards que permetin l'automatització d'un projecte de ML.

[7] https://docs.travis-ci.com/
[8] https://mlflow.org/

1.2 Estructura

En aquest apartat s'explica com s'ha decidit estructurar el treball de final de grau.

El projecte es divideix en quatre parts principals:

1. el marc teòric
2. objectius
3. proposta tècnica
4. conclusions

Primerament en el marc teòric o marc de referència, es recopilen antecedents, investigacions prèvies i consideracions teòriques necessàries per entendre el projecte i el posterior anàlisis dels resultats i formulació de les conclusions. El marc de referència es caracteritza per tenir un llenguatge teòric on es defineix la disciplina la qual pertany al camp de l'estudi escollit.

Tot seguit en els objectius es planteja els requeriments bàsic que ha de complir el treball proposat en el projecte, tant a nivell genèric com específic.

En l'apartat de la proposta tècnica es plasmen els motius pels quals resulta beneficiós realitzar el projecte plantejant i els aspectes tècnics de com s'ha realitzat la implementació per tal d'intentar assolir els objectius.

Finalment, en les conclusions es comparen els resultats obtinguts respecte als objectius inicialment plantejats.

1.3 Marc teòric

1.3.1. El cicle de vida del *Machine Learning*

El cicle de vida o flux de treball del *Machine Learning* (ML), és un procés iteratiu que habitualment se segueix de forma implícita quan es realitza un projecte de *Data Science.*

El cicle de vida defineix certes directrius i bones pràctiques a seguir per a cadascuna de les seves etapes, amb l'objectiu de garantir qualitat, solidesa i consistència estadística en els resultats d'un projecte dins de l'àmbit de la Intel·ligència Artificial (IA) (Geron 2017).

Actualment, existeixen pocs estàndards que cobreixin un cicle complet de desenvolupament per a tots els projectes d'aquesta tipologia, ja que són els mateixos *Data Scientists* d'un equip els encarregats d'adaptar aquest cicle segons les necessitats i objectius del problema. Tanmateix, hi ha un conjunt d'etapes que acostumen a ser comunes en la majoria de projectes. Aquestes fases són:

- Formalització del problema a resoldre.
- Recol·lecta de dades.
- Tractament de dades i *data cleaning.*
- *Feature engineering.*
- Llistat dels algorismes més adients
- Optimització dels hyperparàmetres.
- Desplegament, seguiment i manteniment del sistema (Geron 2017).

Formalització del problema a resoldre

La primera fase tracta de formalitzar el problema a resoldre i, tot i no pertànyer exclusivament al camp de la IA, és una de les més importants en tota mena de projectes. Des del punt de vista de l'enginyeria de requeriments, aquesta etapa es divideix en tres punts diferenciats: la definició de l'abast i dels principals objectius, l'anàlisi del domini d'aplicació i la formalització del problema (Robertson and Robertson 2012).

Definició de l'abast i dels principals objectius

En aquest punt s'identifiquen els límits del projecte, es determinen quins seran els objectius a assolir i es decideixen els factors clau com ara quin és el seu propòsit, les àrees de treball afectades, els usuaris implicats, les restriccions, etc.

Durant aquesta fase s'ha d'evitar cometre equivocacions, ja que això comportaria l'encadenament d'errors des d'un principi i, probablement, duria lloc al fracàs del projecte (Robertson and Robertson 2012).

Anàlisis del domini d'aplicació i formalització del problema

Aquest apartat del cicle de vida té dues finalitats principals: entendre el domini del problema per a determinar una estratègia per afrontar-lo i fixar les mètriques que permetran avaluar si l'objectiu s'ha assolit o no.

Pel que fa a l'enteniment del domini, ens pot ser de gran ajuda a parlar amb el client o entitat que planteja el problema i mirar si existeixen solucions de problemes similars, entre altres.

Així mateix, partint de l'objectiu establert en la fase anterior, s'emmarca el projecte dins l'àmbit de la IA. Aquests tipus de problemes es poden classificar segons el tipus d'entrenament de l'algorisme que s'utilitzarà i segons quin és l'objectiu del problema (Geron 2017; Robertson and Robertson 2012).

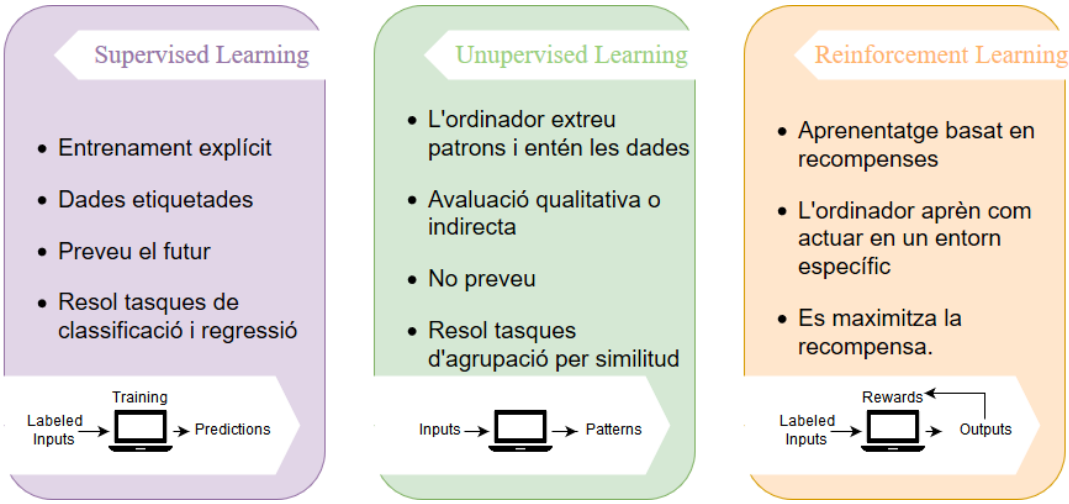

Figura 1: Diferència entre els tipus d'entrenament

Depenent del tipus d'entrenament diferenciem tres tipus principals de problemes.

- Supervisat → Problema on s'utilitzen exemples previs per entrenar l'algorisme, és a dir, les dades que s'utilitzen per entrenar l'algorisme inclouen les solucions desitjades, les quals s'anomenen *labels.* Posteriorment, el model entrenat serà capaç d'assignar una *label* a un registre que no ha vist en la fase d'entrenament.
- Aquest tipus de problema acostuma a dependre d'un gran nombre de dades, que sovint estan etiquetades a mà. Els conjunts de dades etiquetats a mà són molt costosos tant econòmicament com temporalment, especialment quan es requereixen experts del tema per dur a terme la classificació dels diferents registres. Per aquests motius apareix

el supervisat dèbil també anomenat *weak supervision*[9]. La *weak supervision* consisteix en etiquetar instàncies del conjunt d'entrenament més eficientment però amb un etiquetat de més baixa qualitat i a més alt nivell utilitzant heurístiques, restriccions, classificadors entrenats amb poques dades, etc.

- No supervisat → Problema en què les dades utilitzades per entrenar l'algorisme no consten de *labels.*
- *Reinforcment learning* → És un enfocament completament diferent dels dos anteriors. En aquest tipus d'entrenament existeix un agent capaç d'observar el seu entorn i realitzar accions sobre ell. Cada acció pot comportar guanyar o perdre punts i l'objectiu d'aquest agent és maximitzar els punts guanyats (Geron 2017).

Depenent del objectiu del problema diferenciem entre els següents tipus:

- Classificació → S'utilitzen quan el resultat desitjat és una etiqueta discreta. En altres paraules, s'utilitza quan la resposta a l'objectiu recau sobre un conjunt finit de possibilitats.
- Regressió → S'utilitza quan els resultats a predir són un valor continu, és a dir, que la resposta és una quantitat flexible basada en les entrades del problema.
- Recomanació → S'utilitza quan l'objectiu se centra a recomanar un producte nou a un usuari d'acord amb els seus interessos. Aquestes tasques són molt freqüents en plataformes digitals d'àmbits com la música, notícies, llibres, etc.
- Agrupació → S'acostuma a utilitzar en problemes no supervisats, on l'objectiu consisteix a agrupar un conjunt d'instàncies basant-se en una mesura d'igualtat(Geron 2017).

Una vegada emmarcat el problema dins la tipologia dels projectes de l'àmbit de la IA, és aconsellable realitzar un llistat d'estratègies diferents que podrien ser útils per resoldre el problema. Cada estratègia ha de contemplar un conjunt d'algorismes i una mètrica per avaluar si els resultats obtinguts són vàlids o no.

En el món del ML, els algorismes més freqüents són els de classificació i regressió. A continuació s'exposen els mètodes que s'acostumen a utilitzar per realitzar una avaluació sòlida.

[9] http://ai.stanford.edu/blog/weak-supervision/

Avaluació en tasques de regressió

En un problema de regressió no tradicional, l'objectiu és trobar la recta que passi el més a prop possible per tots els punts. Per exemple, si partim d'una distribució de punts com a la Figura 3, obtindríem una recta com la de la Figura 2.

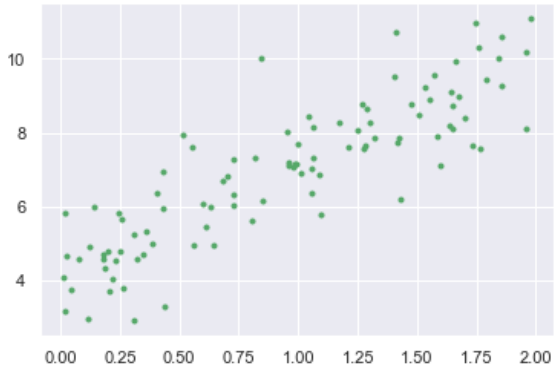

Figura 3: Dispersió de punts

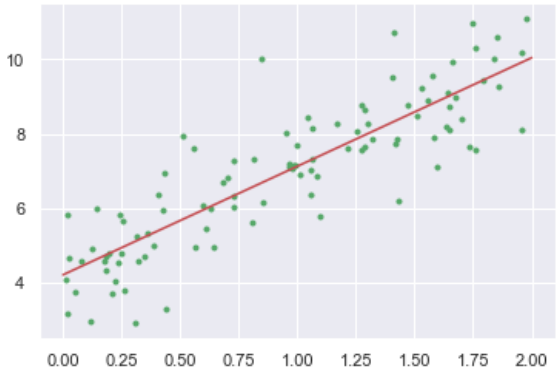

Figura 2: Regressió lineal

Si el nostre objectiu es trobar la línia que s'aproxima més a tots els punts una solució "desesperada" seria traçar una gran quantitat de línies i comprovar quina es la més propera a les dades (Figura 4).

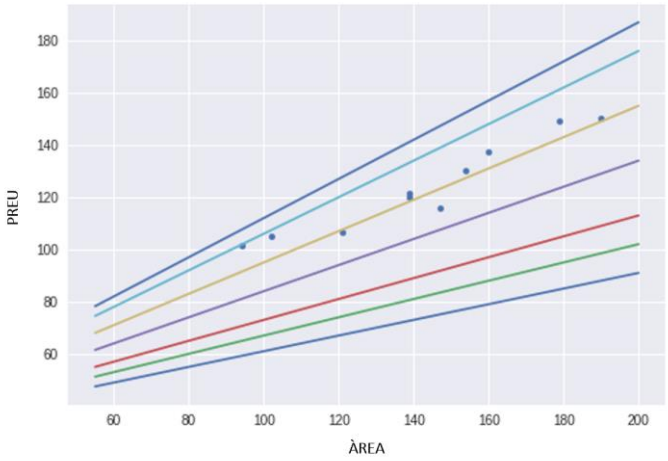

Figura 4

Aquesta estratègia té dos grans problemes:

1. Existeixen infinites línies.
2. No sabem com mesurar si una línia es millor que una altra.

El segon problema es pot afrontar definint una funció de cost. Una funció de cost és un mètode per avaluar com es comporta un algoritme en un conjunt de dades determinat. En aquest cas,

la funció de cost calcula la mitjana de les diferències entre els resultats obtinguts d'una hipòtesis basada en els valors d'entrada i el resultat real.

Així doncs, es defineix una mètrica que permeti calcular la distància entre la recta (els valors predits) i el conjunt de punts objectiu. Per tant, la finalitat serà minimitzar aquest cost. A més aquesta funció de cost també permet realitzar una avaluació sobre el model, ja que com més petit sigui aquest cost millor seran les prediccions del model.

Les funcions d'error més utilitzades són les següents:

- *Root Mean Square Error (RMSE)* → Representa la desviació estàndard de cada punt al seu valor predit.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Equació 1: Root Mean Square Error (RMSE)

- *Mean Absolute Error (MAE)* → És la mitjana de les distàncies absolutes entre el valor predit i el valor observat. El MAE és un valor lineal que penalitza per igual totes les diferències. Per exemple, la diferència entre el 10 i el 0 és el doble que la distància entre 5 i 0, en canvi, aquest fet no passa amb el RMSE.

$$MEA = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Equació 2: Mean Absolute Error (MAE)

El fet de tenir dues possibilitats d'avaluar un model fa plantejar la següent pregunta: Quin és millor? Aquesta pregunta no té una resposta directa, ja que les dues funcions de cost tenen els seus respectius avantatges. Per una banda, el MAE és més simple d'entendre, però per altra banda, el RMSE penalitza més les diferències grans que el MAE, cosa que és una propietat molt interessant en l'avaluació d'un algorisme de regressió.

Tanmateix, tot i ser més complex i tenir una desviació més alta, el RSME continua sent la mètrica predeterminada de la majoria de models, ja que la funció de pèrdua definida en els termes de RSME és més diferenciable i facilita la realització d'operacions matemàtiques (Geron 2017).

Avaluació en tasques de classificació

D'entrada, és necessari entendre la terminologia bàsica utilitzada en l'avaluació en tasques de classificació. Cal diferenciar les quatre categories que sorgiran a partir de la classificació realitzada i de la seva correctesa:

- **True Positives (TP):** Correctament classificats.
- **False Positives (FP)**: Incorrectament classificats.
- **True Negatives (TN)**: Correctament rebutjats.
- **False Negatives (FN)**: Incorrectament rebutjats.

En la Figura 5 podem veure un esquema d'aquesta classificació.

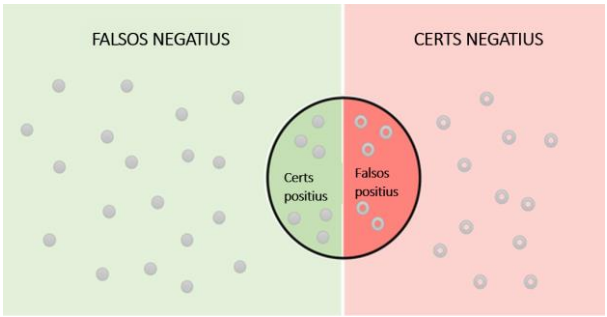

Figura 5: Esquema mètriques d'avaluació de mètodes de classificació

D'aquests quatre valors es poden extreure diverses mètriques, les quals són les més utilitzades en la validació estadística d'algorismes de classificació. Les mètriques esmentades són les següents:

- ***Recall* o Sensibilitat (*True Positive Rate*):** És el nombre d'instàncies correctament identificades com a positives del total de tots els positius.

$$Recall = \frac{TP}{TP + FN}$$

*Equació 3: Recall o Sensibilitat (*True Positive Rate)

- ***Specifity (True Negative Rate):*** És el nombre d'instàncies correctament identificades com a negatius (rebutjats) del total de tots els negatius.

$$Specifity = \frac{TN}{TN + FP}$$

Equació 4: Specifity (True Negative Rate)

- ***Precision:*** És el nombre d'instàncies correctament identificades com a positives del total identificades com a positives.

$$Precision = \frac{TP}{TP + FP}$$

Equació 5: Precision

- ***False Positive Rate (FPR):*** És el nombre d'instàncies incorrectament identificades com a positives del total identificades com a negatives.

$$FPR = \frac{FP}{FP + TP}$$

Equació 6: False Positive Rate (FPR)

- ***False Negative Rate (FNR):*** És el nombre d'instàncies incorrectament identificades com a negatives del total de positives.

$$FNR = \frac{FN}{FN + TP}$$

Equació 7: False Negative Rate (FNR)

- ***Confusion Matrix:*** La matriu de confusió C és una matriu on cada element C_{ij} és igual al nombre d'observacions del grup *i* però classificades en el grup *j*. Per a la classificació binària tenim:
  - C_{00}: TN
  - C_{10}: FN
  - C_{01}: FP
  - C_{11}: TP

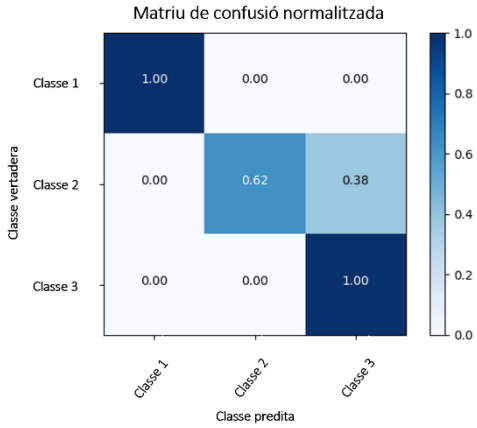

Figura 6: Exemple de Confusion Matrix

- ***F1 Score:*** És la mitjana harmònica de la precisió i la sensibilitat.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Equació 8: F1 Score

- **Accuracy:** És el percentatge del total de les instàncies classificades correctament.

$$Accuracy = \frac{TP + TN}{n} \; on \; n \; és \; el \; nombre \; total \; d'instàncies$$

Equació 9: Accuracy

Observant les definicions de les mètriques estadístiques anteriors, l'objectiu de tot *Data Scientist* seria obtenir els màxims valors possibles tant pel *recall* com per a la precisió. Per desgràcia, aquesta situació no s'acostuma a produir en problemes del món real i s'ha d'escollir entre tenir major *recall* o tenir major precisió, ja que depenent del domini ens interessarà més una opció o l'altra. En la Figura 7 es mostra la corba que exemplifica aquest *trade-off* entre la precisió i la sensibilitat.

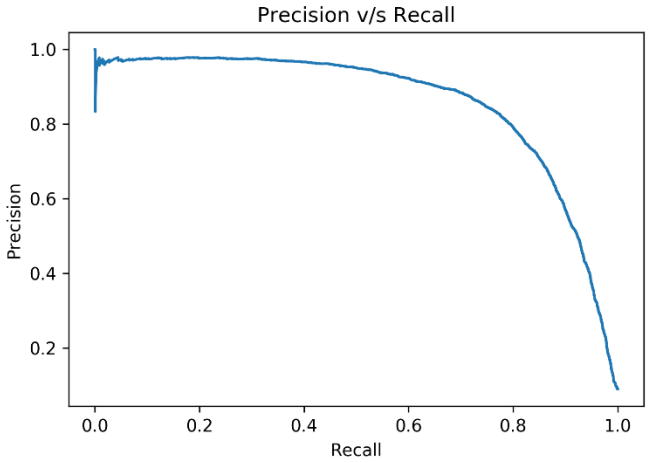

Figura 7: Exemple corba recall vs. precision

Una altra eina molt comuna per avaluar classificadors binaris és la corba *Reciever Operating Characteristic (ROC).* És molt semblant a la corba *Precision vs. Recall,* però en lloc de dibuixar la precisió contra la sensibilitat, dibuixa el *recall* contra el FPR. En la Figura 8 es mostra un exemple de diverses corbes ROC indican si representen un bon model o no (Fawcett 2006; Geron 2017).

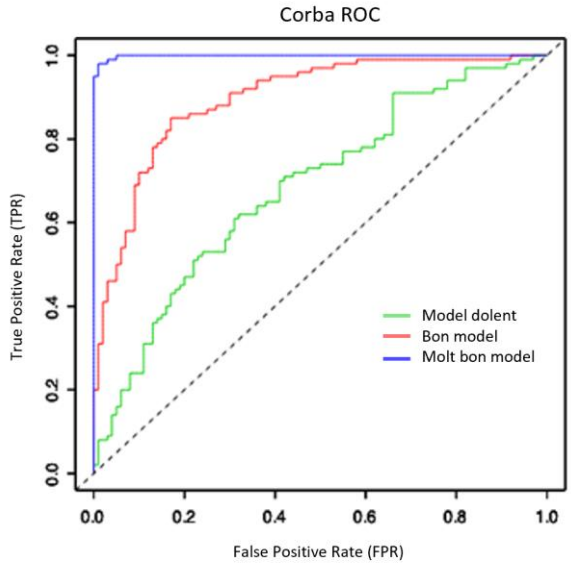

Figura 8: Exemple corba ROC

La corba ROC ens permet avaluar un model de classificació binaria que la seva predicció es basa en calcular la probabilitat que una instància pertany-hi a una classe o no, sent 0 la probabilitat més baixa i 1 la més alta. Utilitzant aquesta probabilitat s'estableix un límit a partir del qual es decideix si una instancia pertany a una determinada classe o no. Per exemple, imaginem que tenim un model que basant-se en els resultats d'una analítica de sang calcula la probabilitat que una persona estigui malalta o no. Suposant que establim el límit a 0.5 direm que una persona esta malalta si la probabilitat resultant del model es superior a 0.5. Per tant, la corba ROC ens permet representar com canvien les prediccions depenent del valor que tingui aquest límit. Una vegada la corba esta dibuixada, sota d'ella, apareix una àrea anomenada *Area Under the Curve (AUC).* L'AUC indica la capacitat que té el model de diferencia elements de diferents classes, de forma que una AUC més gran implica un millor model.(Fawcett 2006)

1.3.2. Recol·lecta de dades

És la fase en la qual es recol·lecten les dades necessàries per entrenar l'algorisme posteriorment.

En primer lloc, cal definir un llistat on s'indiqui les diverses fonts d'extracció de dades, l'autorització necessària per accedir-hi i la quantitat de dades requerides.

En la majoria de casos es tracta amb un volum de dades molt gran, fet que fa necessari utilitzar la tecnologia *Big Data.*

Actualment no existeix un acord genèric sobre la definició de *Big Data* però les explicacions més àmpliament acceptades solen descriure l'expressió en termes dels reptes que implica.

Generalment es descriuen 3 grans reptes del processament d'aquest tipus de dades, anomenats les 3 V's:

- **Volum**: Fa referència a la quantitat massiva de dades. Aquest aspecte ha forçat als científics a dissenyar nous paradigmes de processament i emmagatzematge de dades per tal que el tractament d'aquestes sigui eficient.
- **Velocitat**: Fa referència a la rapidesa en què les dades poden ser recol·lectades.
- **Variabilitat**: Fa referència al problema d'incompatibilitat entre els diferents formats de dades generats (Landset et al. 2015).

En l'àmbit del ML el principal problema que suposa treballar amb *Big Data* és el volum, ja que sovint els conjunts de dades recol·lectats no caben en la memòria d'un ordinador convencional. Quan un conjunt de dades no pot ser emmagatzemat o processat per la memòria d'un ordinador, diem que aquest conjunt és *larger-than-memory.*

Treballar amb aquest tipus de dades és complex i, per tant, és aconsellable utilitzar certes tecnologies que ens facilitin el processament i emmagatzematge d'aquestes. Tanmateix, tot i disposar d'eines que faciliten el flux de treball, és necessari fer un estudi previ d'aspectes com:

- Comprovar la capacitat d'emmagatzematge necessària per poder guardar les dades. En cas que sigui impossible mantenir les dades en un sol ordinador serà necessari distribuir-les en un clúster.
- Crear un entorn de treball capaç de treballar eficientment amb aquest volum de dades.
- Escollir les tecnologies que s'utilitzaran (*Dask*[10]*, Spark*[11]*, Hadoop*[12]*...).*

Una vegada s'ha tingut en compte aquestes consideracions, cal obtenir les dades i convertir-les en un format que s'adapti a l'entorn prèviament configurat. En el procés de conversió de les dades cal tenir en compte que, en el cas d'haver-hi dades sensibles, s'han de seguir els procediments indicats per les directives estatals (LOPD) i europees (GDPR) per tal de garantir l'anonimat de les dades i preservar els usuaris de possibles violacions dels seus drets.

Finalment, se separen les dades obtingudes en tres conjunts:

- ***Training set* (conjunt d'entrenament)** → són les dades que s'utilitzen per entrenar l'algorisme, és a dir, són les dades les quals el model "veu" i aprèn d'elles.

[10] https://dask.org/
[11] https://spark.apache.org/
[12] https://hadoop.apache.org/

- ***Validation set* (conjunt de validació)** → és la mostra de dades utilitzada per a obtenir una avaluació de l'algorisme no esbiaixada sobre el *training set*, i també per a optimitzar els hyperparàmetres, per tant, el model "veu" les dades d'aquest conjunt però no aprèn d'elles.
- ***Test set* (conjunt de test)** → són les dades que s'utilitzen per a fer una avaluació final del model ja entrenat amb els altres dos conjunts, és una bona pràctica deixar aquest conjunt intacte i separat de la resta de dades fins que el model no estigui completament entrenat (Chollet 2017).

La majoria de vegades es divideix les dades en dos conjunts (*traint* i *test*). Després d'això, se separa el conjunt de prova i aleatòriament s'escull un X% del *training set* per tal que sigui el *traint* actual i el (100 – X)% restant sigui el conjunt de validació. Llavors el model és entrenat i avaluat iterativament en aquests conjunts de dades. Aquest procés iteratiu s'anomena *Cross Validation* (CV) (Chollet 2017; Geron 2017).

K-fold CV

Es tracta d'una estratègia per realitzar CV que implica una divisió aleatòria de les instàncies del conjunt d'entrenament en *k* grups, o *folds*, aproximadament de la mateixa mesura. El primer *fold* es tracta com el conjunt de validació, i la resta com al nou *training set*.

A trets generals el procediment que se segueix és el següent:

1. Barrejar les dades aleatòriament.
2. Separar les dades en *k* grups.
3. Per cada grup:
    a. Agafar un grup com a conjunt de validació.
    b. Agafar la resta de grups com a *training set.*
    c. Entrenar el model amb el conjunt d'entrenament i avaluar-lo amb el grup de validació.
    d. Obtenir i guardar els resultats estadístics de l'avaluació i descartar el model.
4. Agrupar els resultats i resumir els resultats

Cada instància és assignada en un únic grup i es queda en aquest *fold* durant tot el procediment. Això significa que cada element és utilitzat una vegada com a validació i *k 1* vegada com a instància d'entrenament (Geron 2017).

Bootstrap CV

És una altra estratègia per a realitzar CV que consisteix a establir *n* iteracions en què a cada iteració es proporciona una divisió aleatòria del conjunt de dades en dos subconjunts: *training set* i *validation set*. A cada iteració s'entrena i s'avalua el model amb els conjunts de dades respectius i, finalment, es fa una agrupació dels resultats obtinguts de cada iteració.

A diferència de *k-folds*, amb *bootstrap*, una instància pot aparèixer vàries vegades o bé cap vegada en el conjunt de validació[13].

[13] http://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/modules/generated/sklearn.cross_validation.Bootstrap.html

1.3.3. Tractament de dades i *data cleaning*

Una vegada ja es disposa de les dades en l'entorn de treball desitjat, aquestes es transformen a un format consistent per tal de poder ser analitzades correctament. Aquest procés s'anomena *data cleaning.*

Aquesta fase és una de les més importants de tot el cicle de desenvolupament, ja que tot i que la majoria de fonaments teòrics del ML estan basats en les prediccions i inferència estadística, normalment s'assumeix que les dades estan en un bon estat. Però, a la pràctica, els Data Scientists passen la majoria de temps preparant i netejant les dades abans de poder aplicar qualsevol operació estadística. És molt poc freqüent que la *raw data* estigui formatada correctament i sense errors.

El *data cleaning* té molt pes en els resultats finals, ja que involucra tècniques d' *imputation* i *outlier handling* que influencien en la tendència de les dades. Per aquest motiu, és una bona pràctica realitzar aquest procediment de forma que sigui reproduïble.

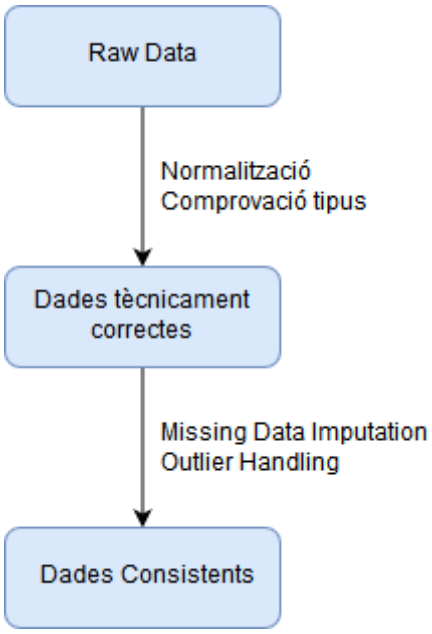

Figura 9: Cicle data cleaning

La Figura 9 mostra un resum d'un procés típic de *data cleaning.* Cada rectangle representa les dades en un estat concret i cada fletxa representa les activitats que s'han de dur a terme per passar al següent estat. El primer estat (*Raw data*) són les dades tal com arriben després de la seva recol·lecta. Aquest tipus de dades acostumen a venir amb tipus incorrectes (per exemple nombres com a *strings*), *encodings* de caràcters desconeguts, etc. La *raw data* és impossible d'utilitzar en programes informàtics sense haver aplicat cap mena de preprocessament. Una vegada aquest s'ha dut a terme, s'obtenen les dades tècnicament correctes. Tanmateix, això no implica que estiguin lliures d'errades. Per exemple, una edat podria ser negativa, una persona menor d'edat podria tenir carnet de cotxe o simplement mancarien algunes dades. Aquestes inconsistències depenen del domini i no poden ser ignorades. Seguidament, l'estat de dades consistent implica que les dades ja estan llestes per la inferència estadística. En aquest punt és on els fonaments teòrics comencen a tenir rellevància.

Per tal de canviar d'estat, és necessari seguir els passos que es detallen a continuació.

De *raw data* a dades tècnicament correctes

L'objectiu principal és obtenir un *data set* a partir de la *raw data.* Un *data set* és una col·lecció de dades que descriuen valors d'atributs (variables) d'un nombre d'elements del món real (instàncies). Entenem dades tècnicament correctes com a un *data set* on cada valor pot ser directament reconegut com a pertanyent d'una variable determinada i que es troba emmagatzemat amb el tipus que representa el valor del domini en el món real.

Amb altres paraules, una variable categòrica ha de ser emmagatzemada com un text i una variable numèrica com un nombre per cada instància, mantenint aquest format consistent en tot el data set.

De dades tècnicament correctes a dades consistents

Les dades consistents són dades tècnicament correctes preparades per l'anàlisi estadística. Són dades en què els *missing values,* els valors especials, els errors i els *outliers* han estat correctament eliminats o substituïts per valors coherents dins el domini de les dades.

La consistència en un conjunt de dades s'obté mantenint la *in-record consistency i la* cross-record consistency. La *in-record* consistency significa que no hi ha informació contradictòria en una instància i la *cross-record consistency* significa que l'agrupació de diferents variables no és conflictiva (de Jonge and van der Loo 2013).

L'eliminació de les dades que provoquen inconsistència enclou dues tasques molt importants: la detecció i correcció d'*outliers* i el tractament de la *missing data.*

Per una banda, tenim la detecció i correcció dels *outliers,* els quals són valors extrems que divergeixen del patró i/o de la tendència general de les dades. En ocasions, aquests valors no ens permeten fer observacions correctes, ja que al ser valors extrems poden canviar i/o ofuscar certes propietats de les dades. Algunes de les causes més comuns que provoquen l'aparició d'*outliers* són*:* errors humans*,* erros dels aparells de mesura o per causes naturals, és a dir, no són producte d'un error. El tractament d'*outliers* és essencial i ajuda a millorar molt el rendiment del model final.

Per altra banda, s'ha de tractar amb la *missing data*. Tractar amb dades buides és molt complex i, de fet, molts experts afirmen que no existeix una forma correcta de fer-ho. Tot i això, s'acostuma a realitzar procediments semblants en la majoria de problemes.

Primerament, cal entendre que no tota la *missing data* sorgeix d'una mateixa causa i, per tant, els seus orígens es poden classificar en tres grups diferents:

Tipus *missing values*	Descripció	Possibles causes
Missing Completely at Random (MCAR)	Perdre dades és completament aleatori i no està influenciat per cap altra dada.	Retirar consentiment de les dades, seguiment interromput...
Missing at Random (MAR)	Es produeix en un temps específic conjuntament amb la insatisfacció dels participants.	Rebuig dades preses, omissió a consciència.
Missing not at Random (MNAR)	Les dades perdudes estan relacionades amb el valor de la mateixa variable.	Per exemple, la *missing* data de la variable IQ, es referiexi a IQs baixos.

Taula 1: Tipus missing values

En els dos primers grups és segur eliminar les dades i despreocupar-se d'elles, en canvi, eliminar les dades en l'altre grup pot provocar que el model no generalitzi bé i, per tant, el més adient seria substituir-les per altres valors. En la Figura 10 podem veure quines són les diferents estratègies per tractar amb la *missing data* (de Jonge and van der Loo 2013).

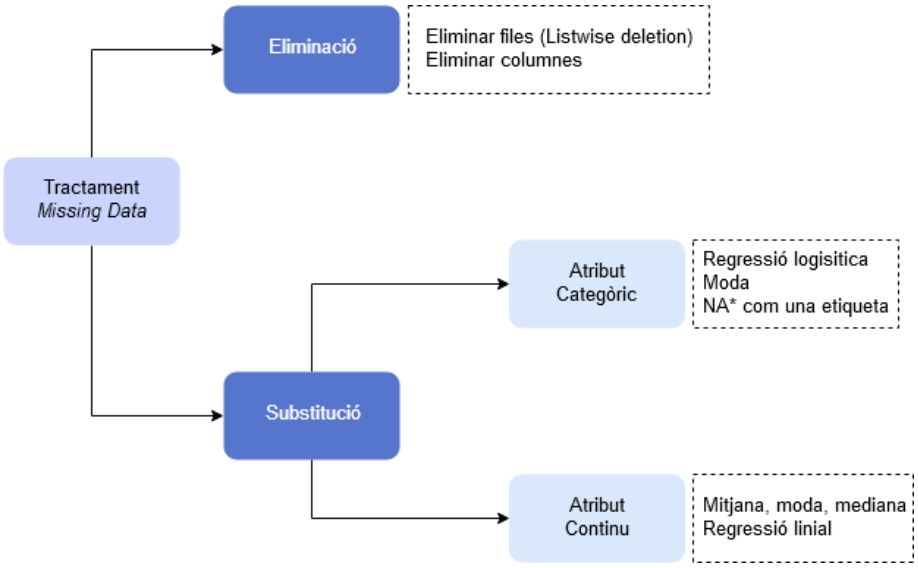

Figura 10: Esquema impute missing data

1.3.4. Feature engineering

Per tal d'entendre el concepte de *feature engineering*, cal conèixer la definició de *feature* en l'àmbit de la ML. Així doncs, una *feature* és un atribut individual, mesurable i típicament representat per una columna en un *dataset.* En un *dataset* genèric de dues dimensions, cada observació es representada per una fila i cada columna representa una *feature* o atribut. En la Figura 11, podem observar una definició gràfica d'una *feature.*

		FEATURES (F)		
		F1	**F2**	**F3**
INSTÀNCIES	**0**	45	49	49
	1	60	62	63
	2	80	82	83
	3	80	99	97
	4	39	52	43

Figura 11: Definició features

Cada fila indica un vector de *features* i tot el conjunt de *features* s'anomena *feature matrix.* Típicament els algorismes de ML treballen amb aquestes matrius numèriques i, per tant, l'objectiu de la majoria de tècniques de *feature engineering* consisteix en convertir les dades d'entrada en una *feature matrix* que potenciï el rendiment de l'algorisme i, a la vegada, li faciliti l'enteniment de les dades.

Les *features* poden ser de dos tipus:

- *Raw* → *Features* que poden obtenir-se directament de les dades inicials.
- *Derived* → *Features* obtingudes mitjançant tècniques *feature engineering,* les quals utilitzen *raw features* per generar-n'hi de noves. Per exemple, calcular l'edat a partir de la *raw feature* "any de naixement" (Chollet 2017).

Feature scaling

Depenent de l'algorisme que es vulgui utilitzar, aquesta fase pot ser omesa. Tot i això, en la majoria de casos, cal escalar les dades de forma que totes les *features* utilitzin la mateixa magnitud. Realitzar aquesta pràctica és recomanable, ja que la majoria d'algorismes utilitzen la distància euclidiana, fet que farà que les *features* amb magnituds més grans tinguin més importància que la resta. Per evitar això, cal portar tots els atributs a el mateix nivell de magnitud (Geron 2017).

1.3.5. Llistat dels algorismes més adequats

En aquest punt del projecte ja s'han realitzat tots els passos necessaris de tractament i adició de dades de forma que ja poden ser tractades i utilitzades per qualsevol algorisme d'aprenentatge automàtic amb la màxima eficiència possible.

Un model o algorisme de ML es descriu com el procés d'aprenentatge d'una funció objectiu anomenada *f,* la qual aconsegueixi convertir les variables d'entrada (x) a una variable de sortida (y) amb un error mínim (ϵ): $y = f(x) + \epsilon$. Aquest aprenentatge es dur a terme amb un procés que s'anomena *model fitting*, amb altres paraules, l'entrenament d'un model amb dades existents és capaç d'ajustar els paràmetres de la funció objectiu.

Tot i això, existeixen altres paràmetres que no són apresos pel algorisme de forma automàtica. Aquests paràmetres expressen certes propietats a més alt nivell dels algorismes, com per exemple, la seva complexitat o la velocitat en què aquests aprendran. Aquests paràmetres s'anomenen hyperparàmetres. Els hyperparàmetres s'acostumin a establir just abans del procés de *model fitting.*

Com els hyperparàmetres no es calculen de forma automàtica és el *Data Scientist,* el qual mitjançant els seus coneixements o altres tècniques d'optimització escull els millors hyperparàmetres per als models entrenats.

Depenent de l'objectiu del problema i les dades, s' escull un conjunt d'algorismes els quals seran entrenats i avaluats amb els respectius conjunts prèviament generats. A continuació s'exposa un llistat dels algorismes més utilitzats depenent de si l'objectiu és de classificació o regressió.

Algorismes més utilitzats

En el món del ML existeix un teorema anomenat *No Free Lunch*[14] . Resumint, el teorema exposa que cap algorisme és millor que un altre, sinó que depenent del problema un serà més adient que un altre. Tot i això, existeixen models que s'utilitzen més que la resta. A continuació s'exposa una llista dels algorismes d'entrenament supervisats més utilitzats.

- **Regressió Lineal (*Linear Regression*)(**Figura 2**)**: Probablement un dels algorismes més coneguts. Resol problemes de regressió descrivint la línia que millor relaciona els valors d'entrada (x) amb els valors de sortida (y) .
- **Regressió Logística (*Logistic Regression*)(**Figura 12**):** Mètode simple per resoldre problemes de classificació binaria. A diferència de la regressió lineal els resultats de la

[14] http://www.no-free-lunch.org

regressió logística són transformats per una funció no lineal anomenada funció logística. Les prediccions fetes per una regressió logística poden ser interpretades com la probabilitat que donada una instancia aquesta pertanyi a una classe o no.

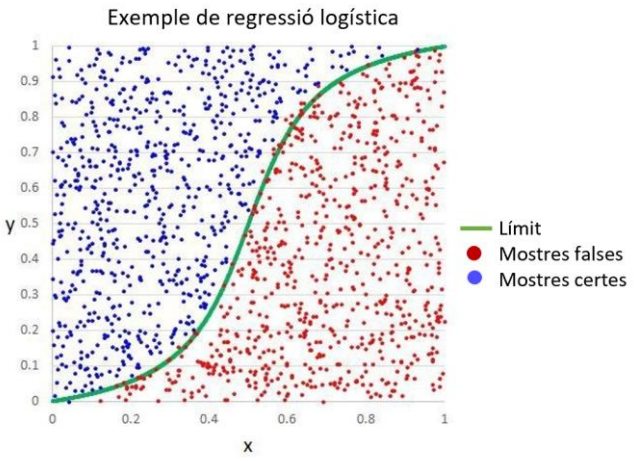

Figura 12: Regressió Logística

- **Arbres de decisió (*Decision trees*)(**Figura 13**)**: Es poden representar com un arbre binari, on cada node representa una instancia d'entrada i un punt de separació depenent del valor d'una *feature* concreta. Les fulles dels arbres contenen la variable resultant (y), la qual s'utilitza per realitzar una predicció. Les prediccions es duen a terme recorrent els nodes de l'arbre fins que s'arriba a una fulla. Els arbres de decisió són capaços de resoldre tant problemes de classificació com de regressió.

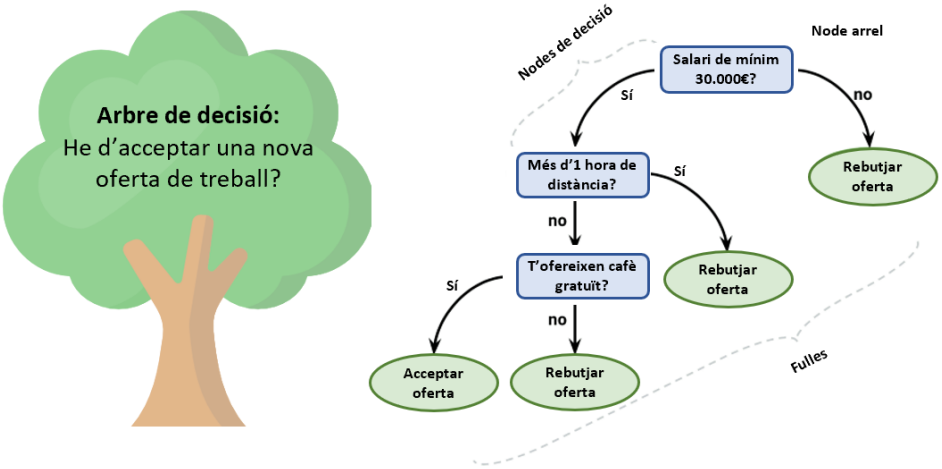

Figura 13: Decision tree

- ***K-Nearest Neighbors**:* És un algorisme molt simple i efectiu. Les prediccions es fan buscant en tot el *training set* les K instancies més similars a la nova instancia del la qual es desitja predir la seva classe i/o valor. Serveix per resoldre problemes tant de

classificació com de regressió. Per exemple, si voler resoldre un problema de classificació la predicció podria ser la classe predominant dels K elements més propers, en canvi en un problema de regressió, el valor resultant seria la mitjana dels K valors més propers.

- ***Suport Vector Machines*** **(SVM)**: És un dels algorismes més eficients i s'utilitza per a resoldre problemes de classificació. Un hiperplà és una línia que separa l' espai on viuen les variables d'entrada. En un SVM es calcula l'hiperplà el qual separa millor els punts que pertanyen a diferents classes. La distancia entre el hiperplà i els punts més propers de cadascuna de les classes (*Support vectors*) s'anomena marge. Els *support vectors* són els únics punts que influeixen en l'entrenament del SVM, el qual té com a objectiu maximitzar el marge.
- ***Bagging & Random Forest****:* El *random forest* és un dels algorismes més potents del ML. És un tipus de *enemble* anomenat *bootstrap aggregation* o *bagging.*

  Bootstrap és un mètode estadístic per estimar una quantitat provinent de diversos subconjunts de les dades. Per exemple, si es desitja calcular la mitjana d'un conjunt molt gran de dades, primerament es generen diversos subconjunts d'aquests, per cada conjunt es calcula la mitja i finalment totes les mitjanes s'agrupen també fent la mitjana. D'aquesta manera s'aconsegueix una millor estimació de la mitjana real.

  En un *random forest* s'aplica *bootstrap* de forma que diversos *decision trees* s'entrenen amb diferents subconjunts de dades i quan es fa una predicció cada arbre la realitza individualment i després s'agrupen de forma que s'estima millor el calor vertader.
- ***Boosting*** **i** ***AdaBoost****:* Igual que el *random forest* també és una tècnica d'assemblatge, que el seu objectiu es crear un classificador fort utilitzant classificadors dèbils. Això s'aconsegueix entrenant un model amb les dades d'entrenament i posteriorment entrenar-ne un altre corregint les mancances del primer i així successivament. *AdaBoost* s'utilitza amb arbres de decisió amb poca profunditat. Després de crear el primer arbre, el rendiment del arbre s'avalua per a cadascuna de les instancies del *training set,* després aquest rendiment s'utilitza per ponderar quanta atenció haurà de prendre el següent arbre a una determinada instancia, de forma que les instancies difícils de predir tindran un pes alt, en canvi les que són fàcils de predir tindran un pes baix. Els diferents models els van creant seqüencialment ajustant les ponderacions en cada entrenament. Una vegada tots els arbres ja s'han entrenat es fan les prediccions en noves dades tenint en compte com es comportava cada arbre en el conjunt d'entrenament, com més precís era l'arbre en el *training set* més pes tindrà en el moment de la predicció.

- **Xarxes Neuronals (*Neural Networks*):** Actualment són considerades el estat de l'art del *deep learning.* Una NN es pot veure com un graf acíclic de capes, i cada capa te una funció de processament de dades específica. Tot i que la majoria de capes tenen estat o *weights* (pesos de la capa), existeixen capes que no en tenen. El conjunt de pesos de cadascuna de les capes formen el coneixement de la xarxa.

  L'entrenament d'una xarxa consisteix a ajustar els pesos de forma que les diferents capes es comportin de la forma mes adequada possible per tal de relacionar els valors de l'entrada amb els valors de sortida. Les xarxes neuronals són capaces de resoldre qualsevol tipus de problema.

1.3.6. Fine-tune del sistema

En aquesta etapa s'han d'utilitzar el màxim nombre de dades possibles.

En el ML hi ha dos tipus de paràmetres, els que un model aprèn duran l'entrenament, per exemple, la bias en una regressió lineal, i els paràmetres de l'algorisme, anomenats hyperparàmetres, els quals han de ser optimitzats a part.

El *fine-tuning* en ML consisteix en fer la regularització i/o optimització dels hyperparàmetres.

Per duu a terme el *fine-tuning* hi ha diverses estratègies, entre les més utilitzades es troben les següents:

- ***Grid Search*** → S'utilitza un llistat dels hyperparàmtres que es volen experimentar, es realitzen totes les combinacions possibles entre ells i, amb cada combinació, s'entrena un model. Dels models resultants s'avalua quin és el millor utilitzant *cross-validation.*
- **Random Search →** Quan l'espai de cerca és molt gran utilitzar *Grid Search* no és aconsellable. El procediment és el mateix que en *Grid Search,* però enlloc de provar totes les combinacions possibles només proba *n* combinacions aleatòries, on *n* es decidit per un *data science.*

Una vegada realitzat el procés d'optimització dels hyperparàmetres s'escull el model que obté millors resultats en l'avaluació.

1.3.7. Desplegament, seguiment i manteniment del sistema

Una vegada disposem d'un bon model capaç d'assolir amb garanties l'objectiu establert, cal posar-lo en producció. En aquesta fase és aconsellable escriure tests unitaris i així automatitzar el procés de prova abans de posar el sistema en producció.

Un cop el sistema està en producció és aconsellable desenvolupar un sistema que comprovi l'estat del model de forma periòdica i que sigui capaç de llançar alertes en cas que es produeixi cap mena d'error.

Pel que fa al manteniment dels models, cal tenir en compte la degradació que aquests poden sofrir. La majoria de models tendeixen a perdre eficiència a mesura que les dades evolucionen i això és a conseqüència que en un món tan digitalitzat, les dades evolucionen molt ràpid i un model entrenat amb dades velles no és capaç de generalitzar bé amb dades actuals. Per detectar aquest tipus de problema és recomanable programar un sistema que periòdicament realitzi un procés d'avaluació al model amb dades actuals.

2. OBJECTIUS

2.1 Objectius generals

Aquest projecte es centra en l'estudi de l'automatització del cicle de vida d'un projecte de ML. Els objectius principals són els següents:

- Anàlisi, disseny i implementació d'un estàndard del cicle de vida de models ML: Analitzar i estudiar diferents projectes de l'àmbit del ML per extreure'n les parts essencials i comuns i utilitzar aquests factors per establir un cicle de desenvolupament prou genèric per a ser utilitzat en tota mena de projectes del món de la IA. A més, per a cadascuna de les etapes definides en el cicle de desenvolupament, establir certes directrius per tal de garantir resultats satisfactoris i estadísticament sòlids.
- Anàlisi, disseny i implementació d'una infraestructura per a l'automatització del desenvolupament de solucions ML: Desenvolupar un conjunt d'eines (*framework)* per tal de facilitar el seguiment i compliment de les directrius del cicle de desenvolupament definit. Aquest *framework* ha de permetre la creació i la integració de projectes de forma que s'adaptin als estàndards establerts.

2.2 Objectius específics

Referent al desenvolupament del *framework* i la infraestructura per a l'execució del cicle de vida els objectius específics són els següents:

- Dotar al *framework* d'un cicle de vida format per components genèrics, de forma que cada desenvolupador pugui crear-ne de nous i adaptar el cicle a les seves necessitats però sempre amb una base comuna.
- Dotar al *framework* d'un mòdul capaç d'executar el cicle de vida localment, és a dir, en un únic ordinador.
- Dotar al *framework* d'un mòdul capaç de distribuir el cicle de vida en un clúster d'ordinadors i accelerar així l'execució del projecte.
- Dotar al *framework* mètodes d'avaluació per validar els resultats obtinguts d'una execució del cicle de vida. A més permetre als desenvolupadors crear mètriques personalitzades.
- Estandarditzar l'accés a les dades mitjançant un accés indexat.
- Estandarditzar la divisió de les dades en múltiples conjunts d'entrenament i avaluació per tal d'obtenir resultats estadísticament sòlids.
- Documentar el cicle de vida establert de forma que qualsevol desenvolupador pugui adaptar nous projectes o bé els ja existents.
- Proporcionar una infraestructura per a executar projectes al núvol amb la finalitat de disposar de millors màquines.

3. DESENVOLUPAMENT

3.1 _____

El llenguatge de programació Python[15] s'estableix com un dels llenguatges més populars per a la informàtica científica. Gràcies a la seva naturalesa interactiva d'alt nivell i a l'ecosistema de biblioteques científiques com ara *scikit-learn* (Pedregosa et al. 2011)*, matplotlib*[16]*, numpy* (Van Der Walt, Colbert, and Varoquaux 2011), *Keras*[17]*...*, és una opció molt atractiva per al desenvolupament algorítmic i l'anàlisi exploratori de dades. Així mateix, com a llengua de propòsit general, no només s'utilitza en entorns acadèmics sinó que s'utilitza cada cop més en la indústria (Stackoverflow 2019).

Actualment, el mòdul més utilitzat és *Scikit-learn (sklearn),* el qual es tracta d'un mòdul de Python que integra un gran nombre d'algorismes actualitzats de ML supervisats i no-supervisats. Aquest paquet se centra a adaptar el ML a no utilitzar un llenguatge de programació de propòsit general (Python). A més, *sklearn* posa molt èmfasi en facilitar l'ús de la llibreria, la bona documentació i la consistència en la seva API.

Gràcies a la consistència de l'API de *Sklearn,* existeixen *frameworks* que estableixen funcionalitats extra sobre un procés habitual de tractament de dades i d'entrenament i avaluació d'algorismes. L'eina més utilitzada en aquest aspecte és *mlflow,* la qual aborda tres objectius principals:

- Realitzar un seguiment d'experiments per a registrar i comparar paràmetres i resultats.
- Empaquetar codi ML de forma reutilitzable i reproduïble per a compartir amb altres Data Scientist o fer transferència a producció.
- Gestionar i implementar models de diverses biblioteques de ML a una varietat de servidors de models i plataformes d'inferència.

Tot algorisme de ML disposa de paràmetres que han de ser modificats per tal que l'entrenament sigui més ràpid i s'obtingui millors resultats. Normalment, escollir aquests paràmetres resulta complex i requereix cert grau de coneixement sobre l'algorisme que s'intenta optimitzar i, per aquest motiu, és freqüent utilitzar tècniques d'optimització automàtica. Per exemple, el mòdul de Python anomenat *hyperopt* (Bergstra, Yamins, and Cox 2013) utilitza tècniques d'optimització bayesianes per tal d'optimitzar una funció, la qual seria equivalent a la funció de *sklearn* que implementa un algorisme de ML.

[15] https://www.python.org/
[16] https://matplotlib.org/
[17] https://github.com/keras-team/keras

Tot i que existeix un gran nombre d'eines de codi lliure que faciliten el flux de treball d'un problema de ML, no n'existeix cap a l'abast de la comunitat que faciliti el seguiment de directrius o que permeti treballar amb un cicle de desenvolupament preestablert. Tenir un cicle comú en tota mena de projectes de ML permetria obtenir un procediment estandarditzat de tal forma que qualsevol projecte seria més simple d'entendre per altres *Data Scientists*.

3.2 Proposta tècnica i contribucions

Tot i que no és necessari l'ús d'un *framework* per a estandarditzar el procés de desenvolupament i/o desenvolupar un producte, es recomana el seu ús per tal d'accelerar i millorar els processos que es duen a terme al llarg del desenvolupament de projectes *software* o projectes de IA.

Un *framework* millora diverses tasques, ja que dóna certes garanties, com ara:

- Garanteix que s'està desenvolupant un producte el qual complirà amb les regles de negoci prèviament establertes.
- Garanteix una estructura de projecte consistent.
- Garanteix que el producte resultat serà fàcilment mantenible i actualitzable al llarg del temps.

A més a més, el framework agilitza certs processos, ja que permet estalviar temps als desenvolupadors o *data scientists* dotant-los d'implementacions genèriques de funcionalitats bàsiques o molt freqüents en l'àrea en la qual s'especialitza, de forma que els programadors poden reutilitzar aquests mòduls genèrics i així es poden concentrar en els requeriments més importants del projecte.

De fet, el principi bàsic d'un *framework* és: "No haver de reinventar la roda", és a dir, no ens hem de centrar a tornar a programar el que algú altre ja ha fet prèviament, sinó que és imprescindible centrar-se el màxim possible en les regles de negoci establertes en la fase d'especificació de requeriments.

A llarg termini un *framework* garanteix estendre el màxim possible la vida d'una aplicació o model.

Malauradament, actualment quan un equip de *data scientist* d'una empresa treballa en un projecte de ML, ho fa tal com el cap de projecte creu oportú, cosa que comporta que només un equip amb aquest cap sigui capaç de realitzar un manteniment i actualització eficaç d'aquest projecte. Per aquest motiu un *framework* proporciona una estructura que permet a qualsevol desenvolupador, hagi participat o no en un projecte, pugui "adoptar-lo" fàcilment i, per tant, mantenir-lo i actualitzar-lo gairebé tan ràpid com ho faria un programador que ha participat activament en el desenvolupament d'aquest.

Per aquests motius, en aquest treball es presenta DriftAI i les contribucions realitzades en aquest per tal de millorar-lo. DriftAI és un projecte de codi lliure desenvolupat a la consultora de

software GFT[18], publicat sota la llicència *AppVerse,* l'objectiu del qual és estandarditzar i facilitar el desenvolupament de models de ML, tot això prioritzant la senzillesa i consistència de la seva API.

3.2.1. Descripció DriftAI

Figura 14: Logotip DriftAI

Com s'especifica anteriorment en el document, el desenvolupament de models de ML és una àrea que actualment no està estandarditzada i cada *data scientist* treballa seguint els seus propis criteris, fet que comporta certs problemes:

- **El tractament de dades**: Aquest tipus de models depenen complement de les dades i la clau és tractar-les correctament. Per exemple, tasques com el control de versions sobre un *dataset*, el *feature engineering,* l'avaluació del model... no es duen a terme d'una forma coherent i correcta.
- **Estructura del projecte**: A tot *data scientist* li agrada estructurar el seu codi font basat en les seves preferències sense seguir cap mena d'estàndard o forma comuna. Això provoca dificultats a l'hora de treballar nombroses persones en un mateix projecte.
- **Entrenament del model**: Aquesta fase implica iterar diverses vegades sobre el conjunt d'entrenament i avaluació fins a arribar a un una configuració del model que compleixi amb els objectius establers.

[18] https://www.gft.com

DriftAI es defineix com un *framework* que ajuda els *data scientists* en el seu dia a dia automatitzant el cicle de desenvolupament d'un projecte de ML. Les principals funcionalitats de DriftAI són:

- Estandarditzar la forma mitjançant la qual s'accedeix a les dades d'entrenament i validació.
- Proporcionar una capa d'abstracció sobre tecnologies ja existents com ara *sklearn*, *pandas,* etc.
- Permetre iterar entre diferents configuracions de models (*Approaches*) comparant-les entre elles i optimitzant els seus hyperparàmetres.
- Proporcionar una estructura comuna per a tots els projectes.
- Reduir el temps de posar en producció un model.

Tal com s'especifica en la definició de DriftAI, a més d'oferir les funcionalitats esmentades, també automatitza el cicle de vida d'un projecte de ML basat en les següents fases:

- Realitzar la càrrega d'un *dataset* des d'un origen (*Datasource*).
- Dividir els *datasets* en diferents subconjunts.
- Realitzar l'entrenament i optimització dels models (distribuïdament o localment).
- Dur a terme l'avaluació dels models.

3.2.2. Ingredients DriftAI

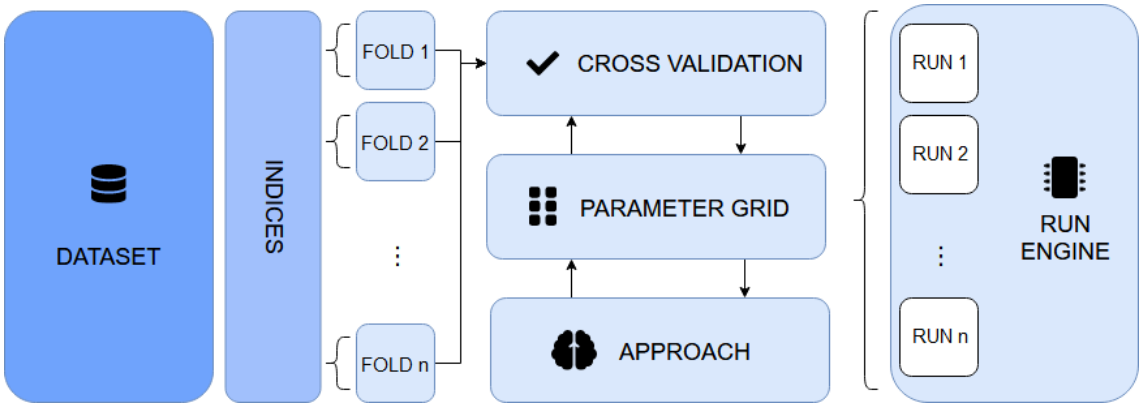

Figura 15: Diagrama DriftAI

En la Figura 15 es poden observar tots els "ingredients" o elements que apareixen en un projecte de DriftAI. A continuació i s'explica cadascun d'aquests elements.

Datasource, Dataset & Indices

Un *dataset* convencional és un simple conjunt de dades com és un fitxer *csv*, un directori amb imatges, etc. DriftAI proporciona una capa d'abstracció sobre els *datasets* convencionals

anomenada *Datasource* o origen de dades. Aquest nom es deu al fet que DriftAI obté les dades d'aquest origen. L'accés al *Datasource* no es realitza d'una forma canònica, sinó que es fa mitjançant índexs (*Indices*), és a dir, que mitjançant identificadors únics per cada instància, siguin numèrics o categòrics, accedeix al *Datasource* i només llegeix les dades que s'identifiquen amb els *indices* corresponents. El conjunt de tots els *indices* és el que, en DriftAI, s'anomena *Dataset.*

L'accés indexat a les dades té diversos avantatges, els més notables dels quals són l'estalvi de memòria i l'eficiència en el tractament de dades.

- **Estalvi de memòria**, ja que només es carreguen les dades sota índexs determinats.
- **Eficiència en el tractament de dades**, ja que moltes vegades per modificar o moure de posició una instància del *dataset*, no és necessari utilitzar tota la instància en si, sinó que simplement amb només una referència és suficient. Per exemple, DriftAI realitza les particions en els *datasets* basant-se només amb l'índex.

Folds, & Subdataset

Tal com s'indica en el marc teòric de l'apartat de recol·lecció de dades, una vegada es disposa d'un conjunt de dades tractat, cal dividir-lo en diferents conjunts per tal d'avaluar els models resultants. Per representar un *dataset* dividit, DriftAI crea un nou objecte anomenat *Subdataset.* Un *Subdataset* conté tots els *folds* (subconjunts de índexs resultants de realitzar una divisió a un *dataset*) i la respectiva informació de quin mètode i paràmetres d'aquest s'han utilitzat per crear-lo.

DriftAI permet la divisió d'un *dataset* un nombre de vegades indefinit utilitzant diversos mètodes. Per exemple, un *dataset* es pot dividir utilitzant diversos valors de *k* utilitzant *k-folds* o inclús dividir el *dataset* utilitzant el mètode *train test split*, el qual divideix el *dataset* en dos conjunts, un d'entrenament i un d'avaluació.

Utilitzar diverses particions d'un mateix conjunt de dades permet obtenir avaluacions més consistents estadísticament.

Parameter Grid & Cross Validation

És l'espai d'hyperparàmetres a avaluar definit per l'usuari. Per definir l'espai, l'usuari ha d'especificar quins hyperparàmetres es volen explorar i en quin rang desitja que es realitzi aquesta exploració. Actualment DriftAI només suporta l'exploració d'hyperparàmetres per Grid Search, és a dir, que donat un rang per cadascun dels paràmetres explora totes les possibles

combinacions. Per tal d'avaluar cada possible combinació, DriftAI utilitza la validació creuada (*Cross Validation),* avaluant així tots els subconjunts d'un *Subdataset* entre ells.

Approach

Un *Approach* és una estratègia que s'utilitza per resoldre el problema d'un *Subdataset.* Per exemple, en un problema de classificació indicarà a quina classe pertany cada instància. Un *Approach* defineix 3 mètodes:

- ***learn:*** Mètode per entrenar el model i retornar aquest ja entrenat.
- ***inference:*** Mètode per predir la label de cada instància i retornar les prediccions.
- ***parameters:*** Mètode on es definirà l'espai d'hyperparàmetres.

Runs & Run Engine

Un *Run* o execució és una possible combinació de paràmetres del *Parameter Grid* juntament amb els resultats de la inferència corresponent. Un *Run* té diversos estats al llarg de l'execució d'un projecte DriftAI. Els tres estats possibles són els següents:

- ***Waiting****:* Pendent d'execució, encara no té resultats emmagatzemats.
- ***Running:*** La *Run* està sent executada.
- ***Finished:*** La *Run* ja s'ha executat i conté els resultats de la inferència corresponent a la combinació de paràmetres corresponents.

Les diverses *Runs* són executades per un *Run Engine,* també anomenat *Runner.* De forma nativa DriftAI proporciona dos *Runners:*

- ***SingleRunner****:* Executa les diferents *Runs* de forma paral·lela en un sol ordinador.
- ***DaskRunner:*** Utilitzant la llibreria *Dask Distributed*[19]*,* executa les *Runs* en un clúster de *DaskRunners.*

Tot i només proporcionar dos tipus de *Runners,* l'usuari pot definir els seus propis de forma senzilla i ràpida.

[19] http://distributed.dask.org/en/latest/

3.2.3. *Workflow* DriftAI

En aquest apartat s'explica el flux de treball que segueix un projecte de DriftAI i com interactuen entre ells tots els "ingredients " esmentats anteriorment.

Per tal de fer l'explicació més amena i tenir un perspectiva més practica de com funciona DriftAI, el seu *workflow* s'explica seguint un exemple on es podran veure tots els beneficis que DriftAI aporta al cicle de vida d'un projecte de ML.

En l'exemple es descriu com resoldre un problema de classificació, més concretament un problema de *single label multiclass classification*, és a dir, existeixen múltiples classes i cada instancia pertany únicament a una classe.

Per realitzar el projecte de classificació s'utilitza un dels *datasets* més clixés del món del ML: el MNIST[20]. El MNIST és una gran base de dades que conté 70.000 números escrits a mà per empleats i estudiants d'un institut americà anomenat *American Census Bureau*.

Figura 16: Dígits MNIST

El objectiu és identificar quin nombre és cadascun dels dígits escrits a mà (Figura 16). Tot i semblar una tasca senzilla, alguns dígits resulten complexes de classificar inclús per l'ull humà (Figura 17).

[20] http://yann.lecun.com/exdb/mnist/

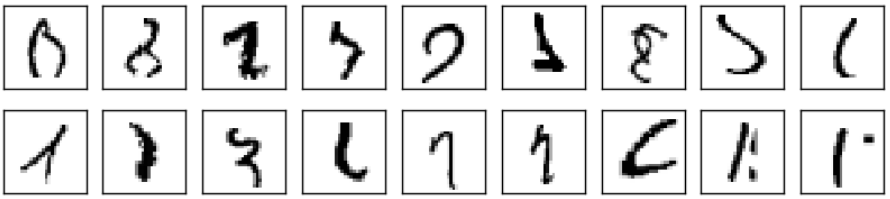

Figura 17: Dígits complexes de classificar

Primerament es procedeix a descarregar el *dataset*[21]. Una vegada ja hem descarregat tots els dígits els organitzarem en un directori anomenat MNIST, que a la vegada contindrà dues carpetes anomenades *train* i *test*, que a la vegada contindran els dígits d'entrenament i avaluació respectivament. Cada imatge s'anomena seguint aquesta nomenclatura: <image_label>_<image_id>.png on <image_label> és l'etiqueta del dígit i <image_id> és un valor autonumèric que comença a 0 i va incrementant, el valor de <image_id> és independent per cada <imge_label> (Arbre de directoris 1). Per exemple, ens podem trobar tant *train/0_0.png* (la primera imatge de l'etiqueta 0) com *train/1_0.png* (La primera imatge de l'etiqueta 1). A la carpeta d'exemples del *repository*[22] de DriftAI podem trobar l'script *download_mnist.py* que permet automatitzar la descarrega i renombrament de les imatges del MNIST seguin la nomenclatura esmentada.

MNIST
test

4_0.png
5_0.png
6_0.png
7_0.png

Arbre de directoris 1: MNIST Dataset

Preparació de l'entorn

Abans de començar a tractar les dades o crear un *Dataset* utilitzant-les, el primer que es desitja fer es crear l'esquema i estructura del projecte. Per facilitar la creació del propi projecte o altres

[21] http://yann.lecun.com/exdb/mnist/index.html
[22] https://github.com/Appverse/DriftAI

objectes, DriftAI proporciona accés a la seva API mitjançant un *Command Line Interface* (CLI). Per crear un projecte amb el CLI cal executar la següent comanda:

```
$ dai  new <project_name>
```

Mitjançant aquesta comanda es crea un arbre de directoris corresponent al projecte de DriftAI anomenat "<project_name>". En el nostre cas volem crear un classificador sobre les dades del MNIST, per tant, anomenarem el projecte "mnist-classifier". Després de fer això l'estructura de directoris resultant és la que es veu en l' Arbre de directoris 2.

```
- CLASSIFIER

driftai.db
```

Arbre de directoris 2: Estructura projecte DriftAI

L'estructura principal del projecte conté dues subcarpetes i el fitxer driftai.db. A continuació s'exposa el que conté cada subcarpeta i fitxer:

- ***approaches***: Aquest directori contindrà un fitxer *python* per cadascun dels *Approches* que s'executaran. Aquest fitxer *python* descriurà els mètodes de *learn* i *inference,* així com l'espai d'hyperparàmetres a explorar.
- ***project_files***: Aquesta carpeta contindrà els arxius necessaris que el usuari cregui necessaris per executar i millorar la comprensió del projecte. Per exemple, pot incloure *jupyter notebooks*[23] amb una exploració prèvia de les dades, referències bibliogràfiques, scripts d'automatització personalitzats, etc...
- ***driftai.db***: És el fitxer més important de tot el projecte, aquest conté tota la informació necessària sobre les configuracions de cadascun dels objectes de DriftAI. Informació com els *indices* d'un *Dataset*, les diferents particions de cada *Subdataset,* els resultats de les diferents execucions, etc. estan emmagatzemats en aquesta base de dades incrustada. Tot i que aquest fitxer pot ser explorat amb un editor de text, ja que és tracta un fitxer formatat amb l'estàndard JSON, no ha de ser modificat manualment sota cap concepte, ja que podria comportar una desconfiguració del projecte.

[23] https://jupyter.org/

Generar *Dataset* indexat

Una vegada estan les dades descarregades, estructurades i el projecte creat cal indexar les imatges del MNIST creant la capa d'abstracció *Dataset.* Per crear el *Dataset* també utilitzarem el CLI i executarem la següent comanda:

```
$ dai add dataset - p <datasource_path>
```

En el cas d'aquest projecte, executarem la comanda amb el <datasource_path> igual a la ruta de la carpeta MNIST on prèviament s'han descarregat les imatges. El resultat esperat de la comanda és:

```
Dataset with id MNIST created
```

Això significa que de forma automàtica s'ha creat un nou registre de *Dataset* juntament amb els seus índexs dins el fitxer driftai.db. D'ara en endavant, per referenciar a aquest nou conjunt d'*indices* s'utilitzarà el seu respectiu identificador, en aquest cas "MNIST".

Abans de programar cap model s'haurà de decidir quina estratègia d'entrenament i avaluació seguirem per tal de trobar la millor combinació d'hyperparàmetres possible. Per exemple, en aquest cas procedirem a dividir el *Dataset* utilitzant *K-fold Cross Validation* amb K = 5. Per crear el *Subdataset* utilitzarem la següent comanda:

```
$ dai generate subdataset MNIST -- method k_fold -- by 5
```

Amb aquesta instrucció s'ha generat un *Subdataset* amb l'identificador "MNIST_k_fold_5", també obtingut a partir de la sortida de l'execució de la comanda. Aquesta partició està formada per 5 conjunts on cada conjunt estarà format per 4 subconjunts d'entrenament i un restant d'avaluació (Figura 18).

Igual que en el cas del *Dataset,* d'ara en endavant ens referenciarem a aquest nou *Subdataset* mitjançant el seu identificador.

Tota la informació referent al últim *Subdataset* creat serà enregistrada a la base de dades incrustada. Aquesta informació permetrà a DriftAI injectar les dades provinents d'un origen de dades de forma transparent als mètodes de *learn* i *inference* dels diversos *Approaches,* els quals seran definits sobre el *Subdataset* MNIST_k_fold_5.

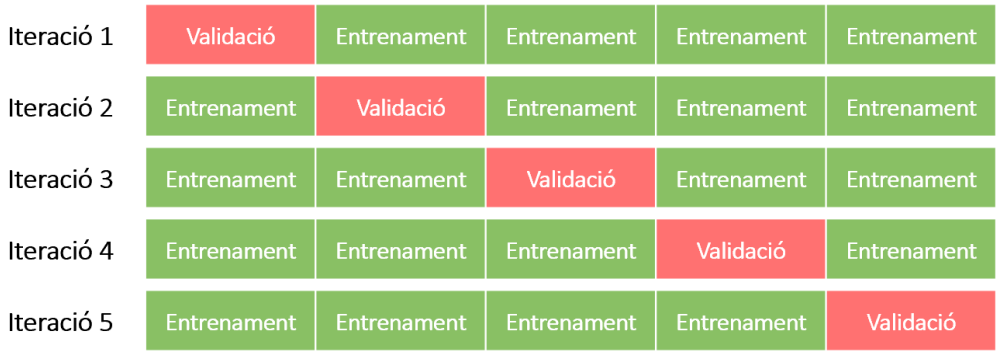

Figura 18: Exemplificació k-folds per k = 5

Creació i execució del *Approach*

A continuació definirem un *RunnableApproach.* Un *RunnableApproach* es el nom que es dóna als scripts que permeten l'execució d'un *Approach* i que, per tant, s'encarreguen de gestionar les diferents *Runs* i escollir el *RunEngine* que s'utilitzarà per executar-les. Un *RunnableApproach* sorgeix de la relació de dos objectes de DriftAI:

- Un *Subdataset*: Conté els conjunts de dades, tant d'entrenament com d'avaluació, que seran injectades durant l'execució d'un *RunnableApproach.*
- Un *Approach*: Conté metadades, *Runs* i resultats de l'execució d'un *RunnableApproach*.

Per tal de crear un *Approach* i vincular-lo amb un *Subdataset* cal executar la següent comanda:

```
$ dai  generate approach random_forest  -- subdataset MNIST_k_fold_5
```

```
- CLASSIFIER
approaches


driftai.db
```

Arbre de directoris 3: Estructura projecte DriftAI amb un approach creat

"random_forest" és el nom o identificador que li proporcionem al nou *Approach,* en aquest cas l'anomenem així, ja que per resoldre el problema plantejat utilitzarem l'algorisme de classificació *RandomForest*. Una vegada ja executada la comanda es pot observar com s'ha creat un nou fitxer *python* dins la carpeta *approaches* anomenat igual que l'identificador que s'ha proporcionat a l'*Approach*, en aquest cas "random_forest" (Arbre de directoris 3).

Dins del fitxer random_forest.py podem observar que s'ha creat un esquelet bàsic (Fragment de Codi 1) d'un *RunnableApproach* on només cal complimentar els mètodes que s'exposen a continuació:

- ***Learn,* també anomenat *train***: Dins aquest mètode s'ha de dur a terme l'entrenament de l'algorisme. En aquest mètode no estem lligats a cap tecnologia sinó que podem utilitzar el programari estiguem més còmodes, com ara sklearn, keras, etc. En aquest exemple utilitzarem l'algorisme *RandomForestClassifier*[24] que proporciona l'API de *scikit-learn.*
- **Inference**: Dins aquest mètode realitzarem les prediccions sobre el test set.
- ***parameters***: En aquesta propietat definirem el llistat de hyperparàmetres juntament amb el seu rang a explorar.

```
from driftai import RunnableApproach
from driftai.run import single_run

@single_run
class RandomForestApproach(RunnableApproach):

    @property
    def parameters(self):
        """
        Declare your parameters here
        """
        return []

    def learn(self, data, parameters):
        """
        Define, train and return your model here
        """
        return None # Return a trained model

    def inference(self, model, data):
        """
        Use the injected model to make predictions with the data
        """
        return None  # Return the prediction
```

Fragment de Codi 1: Esquelet bàsic RunnableApproach

En l'esquelet d'un *RunnableApproach* cal destacar diversos punts molt importants:

[24] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

- @single_run: Per defecte un *Approach* esta anotat pel decorador @single_run el qual significa que automàticament s'injectarà una instància de *SingleRunner* a , i per tant, les diferents *Runs* s'executaran de forma paral·lela en local. DriftAI ofereix més decoradors per injectar de forma automàtica altres *RunEngines.* Per conèixer altres *Runners* es recomana consultar la documentació del mòdul driftai .run [25].
- RandomForestApproach: El nom de la classe que hereta de la classe abstracta *RunnableApproach* segueix la següent convenció: <id_approach> en PascalCase26 + "Approach". Es recomana no modificar el seu nom, ja que internament DriftAI l'utilitza per detectar de forma automàtica a quin *Approach* i a quin *Subdataset* cal referenciar en el moment de l'execució.

Ja només falta complimentar els mètodes per executar el nostre *Approach*.

Primerament descriurem els hyperparàmetres a explorar i el seu rang. DriftAI disposa d'un mòdul que facilita la declaració dels diferents paràmetres del algorisme. En aquest cas com treballem amb el algorisme *RandomForest* de *sklearn* cal mirar la seva documentació per conèixer el tipus i nom de tots els possibles paràmetres[21]. En aquest exemple explorarem els següents paràmetres:

- ***criterion:*** És un paràmetre de tipus categòric. Indica com es calcula la impuresa dels diferents nodes dels arbres de decisió.
- ***N_estimators:*** És un paràmetre de tipus enter. Indica el nombre de arbres de decisió que s'utilitzaran per fer l'*ensemble*.
- ***max_depth*****:** És un paràmetre de tipus enter: Indica la màxima profunditat que tindrà cada arbre.

[25] https://driftai.readthedocs.io/en/latest/api/driftai.run.html
[26] http://wiki.c2.com/?PascalCase

Per a definir aquest espai d'hyperparàmetres s'utilitza el Fragment de Codi 2.

```
from driftai .parameters import CategoricalParameter, IntParameter
...
@property
def parameters (self ):
    """
    Declare your parameters here
    """
    return [
        CategoricalParameter( 'criterion' , [ 'gini' , 'entropy' ]),
        IntParameter( 'n_estimators' , initial =10, limit =100, step =10),
        IntParameter( 'max_depth' , initial =5, limit =15, step =3)
    ]
...
```

Fragment de Codi 2: Definició espai hyperparàmetres

DriftAI proporciona diferents objectes per representar diferents tipus de paràmetres, a part dels que es poden veure en el Fragment de Codi 2, també existeixen els tipus *FloatParameter* i *BoolParameter*, per representar paràmetres reals i booleans respectivament.

Tot seguit definim els mètodes d'entrenament i inferència (Fragment de Codi 3).

```
from sklearn.ensemble import RandomForestClassifier
...
def learn (self , data , parameters ):
    """
    Define, train and return your model here
    """
    model = RandomForestClassifier(
                                    max_depth=parameters.get( 'max_depth' ),
                                    n_estimators =parameters.get( 'n_estimators' ),
                                    criterion =parameters.get( 'criterion' ))
    model.fit( X=data[ 'X' ], y=data[ 'y' ])
    return model

def inference (self , model, data ):
    """
    Use the injected model to make predictions with the data
    """
    return model.predict(data[ 'X' ])
...
```

Fragment de Codi 3: Declaració mètodes learn i inference amb sklearn

En aquest cas com els paràmetres de DriftAI s'han definit utilitzant els mateixos noms que utilitza *sklearn* es recomanable simplificar el mètode *learn* (Fragment de Codi 4), i així complir el principi

SOLID (C. Martin Robert 2000) *Open Close*[27] de forma que no serà necessari modificar el mètode *learn* en cas d'actualitzar l'espai d'hyperparàmetres.

```
...
def learn (self , data , parameters ):
    """
    Define, train and return your model here
    """
    model = RandomForestClassifier(**parameters)
    model.fit( X=data[ 'X' ], y=data[ 'y' ])
    return model
.. .
```

Fragment de Codi 4: Mètode learn simplificat

Finalment, quan ja hem decidit com entrenar el nostre model i fer prediccions utilitzant-lo es el moment d'executar l'*Approach* i començar amb la cerca d'hyperparàmetres. Per fer-ho utilitzarem la següent comanda del CLI:

```
$ dai run random_fores t
```

On *random_forest* és el nom de l'script o l'identificador de l'*Approach*. Per tal de conèixer l'estat de l'execució i el temps restant aproximat, DriftAI proporciona una barra de progrés (Figura 19).

```
Loading approach data
Removing previous runs...
Generating runs...
Saving new runs...
Running...
[====>---------------------------------] 10 %
```

Figura 19: Output execució Approach

Abans de continuar i veure com s'avaluen els resultats obtinguts, es detallen els procediments que segueix DriftAI per tal de dur a terme una execució.

Primerament, quan s'executa l'*Approach* es generen automàticament totes les execucions (Runs), una *Run* per cadascuna de les possibles combinacions dels hyperparàmetres i aquestes s'inicialitzen amb l'estat '*waiting*', és a dir, que estan pendents d'execució. Quan totes les *Runs* ja estan generades DriftAI les executa progressivament. Quan una *Run* està sent executada

[27] https://en.wikipedia.org/wiki/Open%E2%80%93closed_principle

aquesta canvia el seu estat a *running* fins que acaba, moment en que s'emmagatzemen les prediccions com a resultats i el seu estat canvia a *finished.* Mantenir l'estat de les *Runs* permet parar-les i reprendre-les quan es vulgui. Per reprendre les execucions en l'últim punt que es van deixar cal afegir el *flag resume* a la comanda *run*:

```
$ dai run random_forest -- resume
```

Aquest *flag* evita que es tornin a generar les *Runs* i que només s'executin les *Runs* que tenen l'estat *waiting.*

Explorar resultats

Després de l'execució ja es disposa de tots els resultats. Amb aquests resultats DriftAI es capaç de calcular diverses mètriques. Algunes de les mètriques que DriftAI ofereix per defecte són: el *recall,* la *precision,* l'*accuracy,* el *mae* i el *mse.* Per calcular les mètriques s'utilitza la següent comanda.

```
$ opt evaluate random_forest - m recall - m precision - m f1
```

La comanda *evaluate* permet afegir tantes mètriques com l'usuari cregui oportú per tal de tenir una avaluació consistent del seu model. Una vegada DriftAI hagi calculat totes les mètriques d'avaluació per a cadascuna de les *Runs,* s'exportarà un fitxer csv anomenat *<Approach_id>_evaluation.csv*, en aquest cas *random_forest_evaluation.csv*. Cada fila d'aquest fitxer correspon a una única *Run*, la qual estarà composta per els seus hyperparàmetres, els resultats i cadascuna de les mètriques especificades en la comanda anterior.

Un cop el fitxer de les avaluacions s'ha generat podem explorar-lo en un *jupyter notebook* amb l'ajuda de la llibreria *pandas.* En les Figures 22 i 23 es mostra un exemple de com extreure els hyperparàmetres que maximitzen el *f1-score.*

Explore Random Forest Approach Results

```
import pandas as pd
```

Read the csv file

```
df = pd.read_csv('random_forest_evaluation.csv')
df.head()
```

	criterion	f1	max_depth	n_estimators	precision	recall	subdataset_set	y_pred	y_true
0	gini	0.797558	5	10	0.801568	0.793588	A	['0', '0', '0', '0', '0', '0', '0', '0', ...	['0', '0', '0', '0', '0', '0', '0', '0', ...
1	gini	0.809645	5	10	0.813962	0.805373	B	['0', '0', '0', '0', '0', '0', '0', '8', ...	['0', '0', '0', '0', '0', '0', '0', '0', ...
2	gini	0.808066	5	10	0.811784	0.804382	C	['0', '0', '0', '0', '0', '0', '0', '8', '6', ...	['0', '0', '0', '0', '0', '0', '0', '0', ...
3	gini	0.810321	5	10	0.814140	0.806537	D	['0', '0', '0', '0', '0', '6', '0', '0', '0', ...	['0', '0', '0', '0', '0', '0', '0', '0', ...
4	gini	0.810518	5	10	0.814796	0.806284	E	['0', '8', '0', '0', '0', '3', '0', '0', '0', ...	['0', '0', '0', '0', '0', '0', '0', '0', ...

Figura 20: Llegir els resultats de l'avaluació

Find the parameters which maximize f1-score

Join the metrics of each k-fold split using the mean.

```
parameters = ['max_depth', 'n_estimators', 'criterion']
```

```
metrics_summary = df.groupby(parameters, as_index=False).mean()
```

Find the index corresponding to the max f1 score.

```
max_f1_idx = metrics_summary.f1.idxmax()
```

Finally show the parameters

```
metrics_summary.iloc[max_f1_idx][parameters]
```

```
max_depth            14
n_estimators         90
criterion       entropy
```

Figura 21: Trobar els paràmetres que maximitzen el f1 score

3.2.4. *Machine Learning Continuous Integration* (MlCi)

MlCi és una *Platform as a Service* (PaaS), és a dir, és un servei de *cloud computing* que té l'objectiu d'oferir un ordinador, una plataforma o bé un servidor que permeti als clients desenvolupar, executar i organitzar aplicacions sense tenir en compte la complexitat que comporta cada tasca. El codi font de MlCi[28] està publicat sota la llicència GPL v3[29].

Així doncs, MlCi permet dur a terme l'execució d'un projecte DriftAI al *cloud* quan es detecta un nou canvi a un dipòsit de control de versions, en aquest cas en un *repository* de GitHub[30]. Per tal de crear el projecte i executar el cicle de vida complet que defineix DriftAI, MlCi necessita un fitxer de configuració, ubicat a l'arrel del dipòsit de GitHub, on es defineix de forma declarativa com crear cadascun dels "ingredients" que formaran part d'aquest cicle. Els elements de DriftAI que es defineixen en el fitxer de configuració són els següents:

- Els *Datasets,* juntament amb el seu *Datasource.*
- Els *SubDatasets,* indicant sobre quin *Dataset* es realitzaran les particions i amb quin mètode.
- Els *approaches,* conjuntament amb el seu script de *python* anomenat *RunnableApproach*.
- Les mètriques que s'utilitzaran per avaluar els millors hyperparàmetres.

Fitxer de configuració

MlCi permet configurar el seu comportament i llistar tots els *Datasets, Subdatasets i Approaches* que es desitgin executar mitjançant un fitxer de configuració, en format YAML i ubicat a l'arrel del dipòsit de GitHub, anomenat ***ml-ci.yml***.

Per al fitxer de configuració s'ha escollit l'estàndard YAML[31] pels motius següents:

- És llegible per les persones.
- És molt lleuger i ràpid d'escriure (Sense els símbols { , } ").
- No permet tenir claus duplicades (JSON sí que ho permet).
- Permet comentaris amb el caràcter #.

[28] https://github.com/Guillem96/ml-ci
[29] https://www.gnu.org/licenses/gpl-3.0.html
[30] https://github.com/
[31] https://yaml.org/

A continuació es mostra un exemple complet de com configurar un projecte de MlCi i executar-lo al núvol.

Exemple MlCi

En aquest exemple s'utilitza el *Housing dataset.* Aquest conjunt de dades conté informació sobre diferents cases de California i l'objectiu és preveure el valor de les cases depenent de la seva ubicació, nombre d'habitacions, etc. Més concretament es tracta d'un problema de regressió on la label és el preu. Es pot obtenir el conjunt de dades a través de l'enllaç referenciat a peu de pàgina[32].

Primerament inicialitzarem un projecte git amb el nom de *housing-project*.

```
$ mkdir housing    - project && cd housing      - project && git init
```

Ja que MlCi treballa amb GitHub, afegirem com a origen un dipòsit que prèviament haurem creat.

```
$ git remote add origin          https://github.com/Guillem96/housing          - projectg
```

A l'arrel de la carpeta *housing-project* afegirem els fitxers que utilitzarà el nostre projecte. En aquest cas afegirem els següents fitxers:

- *ml-ci.yml*: Fitxer de configuració de MlCi.
- *data_preprocessor.py*: Fitxer que defineix una *pipeline* de *sklearn* per preprocessar i tractar les dades.
- *custom_transformers.py*: Fitxer *python* on es defineixen els diferents components que formaran part de la *pipeline* de preprocessat de dades.
- *dt_regressor.py*: *RunnableApproach* de DriftAI que utilitza un *Decision Tree* per resoldre el problema.
- *linear_regression.py: RunnableApproach* de DriftAI que utilitza una regressió lineal per resoldre el problema.

Seguidament complimentarem el fitxer de configuració tal com es mostra en el Fragment de codi 6. El fitxer de configuració està format per 4 blocs principals:

- before: Bloc on es defineix la llista de les comandes que s'executaran abans de generar i executar el projecte de DriftAI. Les comandes han de ser natives d'un sistema operatiu

[32] https://raw.githubusercontent.com/ageron/handson-ml/master/datasets/housing/housing.csv

Linux, ja que aquest és el sistema operatiu del qual disposen les màquines que ofereix MlCi.

- datasets: Bloc on es defineix la llista de tots els datasets, juntament amb el seu *Datasource*, que formaran part del projecte de DriftAI. Per cadascun dels elements d'aquesta llista cal especificar la ruta d'origen de les dades sota la clau *path*. En cas que la ruta faci referència a un fitxer csv, opcionalment es pot especificar la columna que representa la *label*. Altrament, si es tracta d'un directori de fitxers, és necessari especificar el tipus de dades que conté, mapejat per la clau *dtype*, com ara imatges, i opcionalment el patró de parseig, relacionat amb la clau *parsing-pattern,* que s'utilitzarà per llegir els fitxers continguts en el directori juntament amb la seva classe. Per exemple, si disposem d'un directori com el que es veu en
- Arbre de directoris 4 definirem un *dataset* com el descrit al Fragment de codi 5.

```
- 10
dog

plane

truck
    1.png
    2.png
    3.jpg
```

Arbre de directoris 4: CIFAR-10 Dataset

```
datasets :

  - path : CIFAR-10
    dtype : img
    parsing - pattern : "{class}/{}.png|jpg "
```

Fragment de codi 5: Directori d'imatges MlCi

- subdatasets: Bloc on es defineix el llistat de totes les particions que es realitzaran sobre un *Dataset* especificat. Per declarar un *SubDataset* cal especificar l'identificador del Dataset original, el mètode que s'utilitzarà per a realitzar les particions juntament amb els seus paràmetres. Per exemple, si s'escull el mètode *k_fold* es obligatori indicar la *k,* en canvi, si s'escull el mètode *train_test*, caldrà indicar el *train_size* amb un valor entre 0 i 1.

- approaches: Bloc on es defineix la llista d'*apporaches* que s'executaran dins el projecte de DriftAI. Per cadascun dels *approaches* cal indicar el *SubDataset* que s'utilitzarà, de forma que les dades puguin ser injectades de forma automàtica, les mètriques per

avaluar les inferències, i el nom, el qual correspon al nom del fitxer *python* que conté el *RunnableApproach.*

```
project : housing - project
before :
  - wget - O raw_housing.csv <housing - dataset - link>
  - python data_preprocessor.py
datasets :
  - path : housing.csv
    label : median_house_value
subdatasets :
  - from : housing
    method: k_fold
    k: 5
    id : housing - sbds
approaches :
  - name linear_regression
    subdataset : housing - sbds
    metrics :
      - mse
      - rmse
  - name dt_regressor
    subdataset : housing - sbds
    metrics :
      - mse
```

Fragment de codi 6: Exemple fitxer de configuració MlCi

En aquest apart no es comentaran la resta de fitxers que formen part del dipòsit *housing-project*, ja que no formen part de l'abast del projecte (*custom_transformers.py*, *custom_pipeline.py*) o bé ja s'han explicat en apartats anteriors, com és el cas dels dos *RunnableApproaches.*

Tot seguit versionem els canvis realitzats en local i els afegirem a l'origen de GitHub mitjançant la comanda *push.*

```
$ git add .
$ git commit -
$ git push origin master
```

Una vegada disposem del projecte a GitHub per integrar-hi MlCi navegarem fins al seu portal web[33] i iniciarem sessió mitjançant el nostre compte de GitHub (Figura 22).

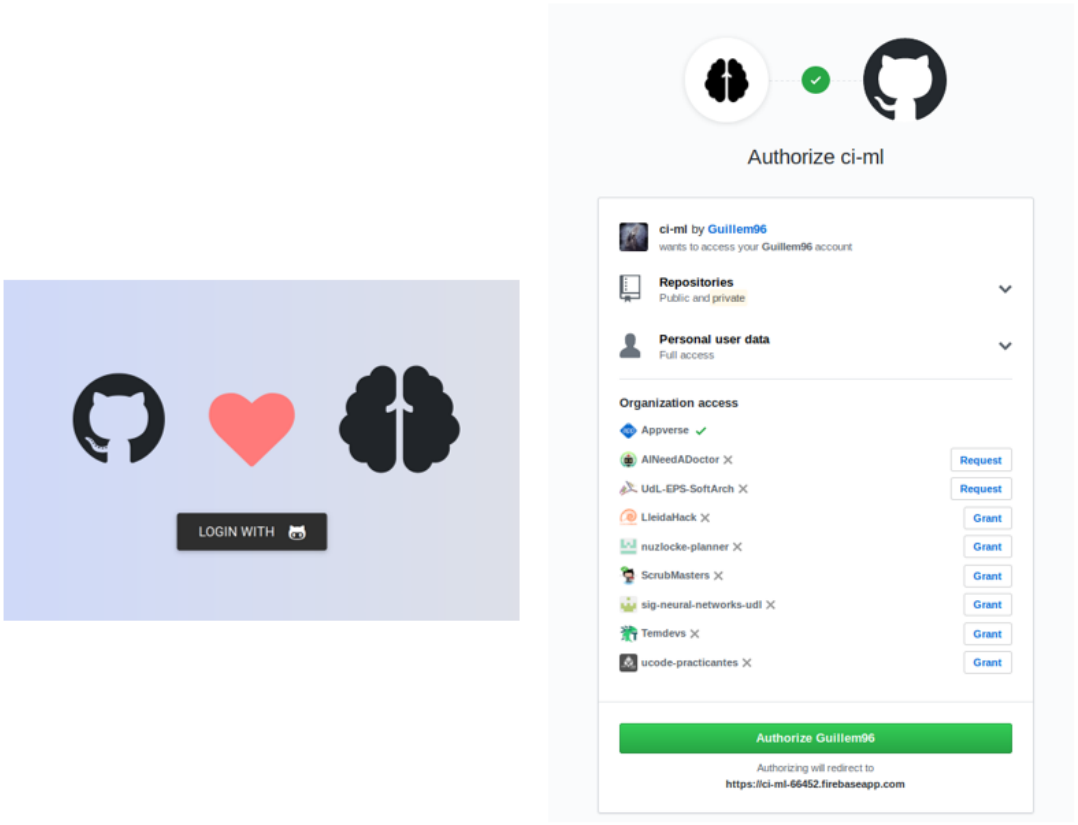

Figura 22: Iniciar sessió amb GitHub en MlCi

Una cop ens haguem registrat utilitzant la nostra conta de GitHub afegirem un *TrackedRepository* que referenciarà el projecte amb el qual em estat treballant (Figura 23).

[33] http://ml-ci.herokuapp.com

Figura 23: Integrar MlCi a un dipòsit de GitHub

Un cop integrat el dipòsit, a MlCi es crea una nova entitat anomenada *TrackedRepository*, nom que prové de la frase *keep track of something*, és a dir, seguir la pista a alguna cosa (en aquest cas, seguir la pista a un depòsit).

Al *dashboard* de MlCi podem observar cadascun dels *TrackedRepositories* juntament amb el seu històric d'execucions (Figura 24).

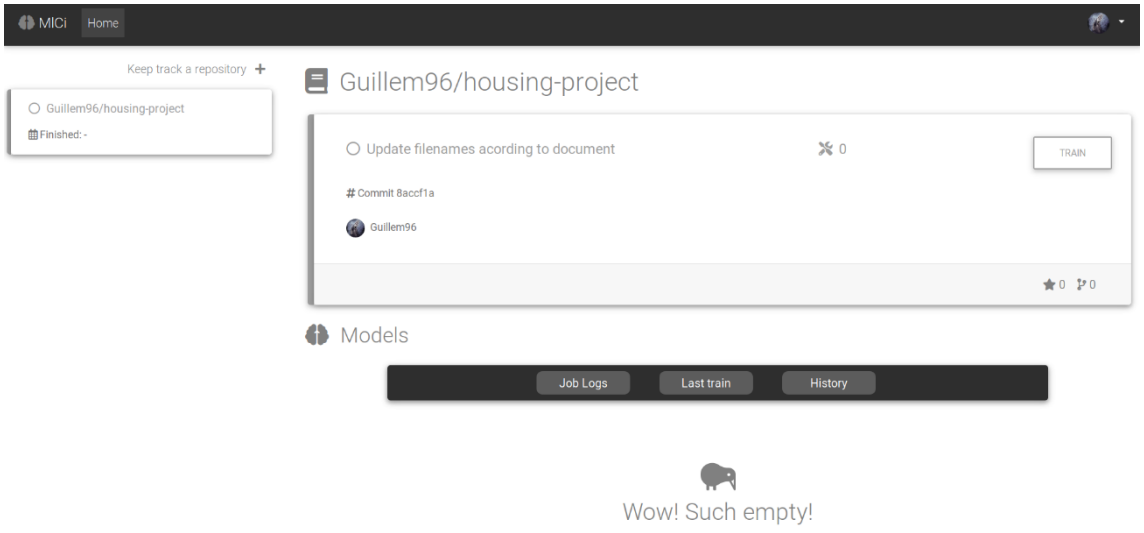

Figura 24: Dashboard MlCi

Com es pot observar, el projecte està pendent d'execució, representat amb els marcs de color gris, ja que no s'ha detectat cap *commit* nou des de la integració de MlCi. Per tal de forçar una execució podem prémer el botó *Train.* Tot seguit, veurem com l'estat del *TrackedRepository* canvia a *TRAINING* (Marcs de color groc). Durant el procés d'entrenament podem observar, a temps real, els *logs* (Figura 25) i l'estat dels diferents *RunnableApproaches* (Figura 26), veient així com evoluciona l'execució del projecte de DriftAI.

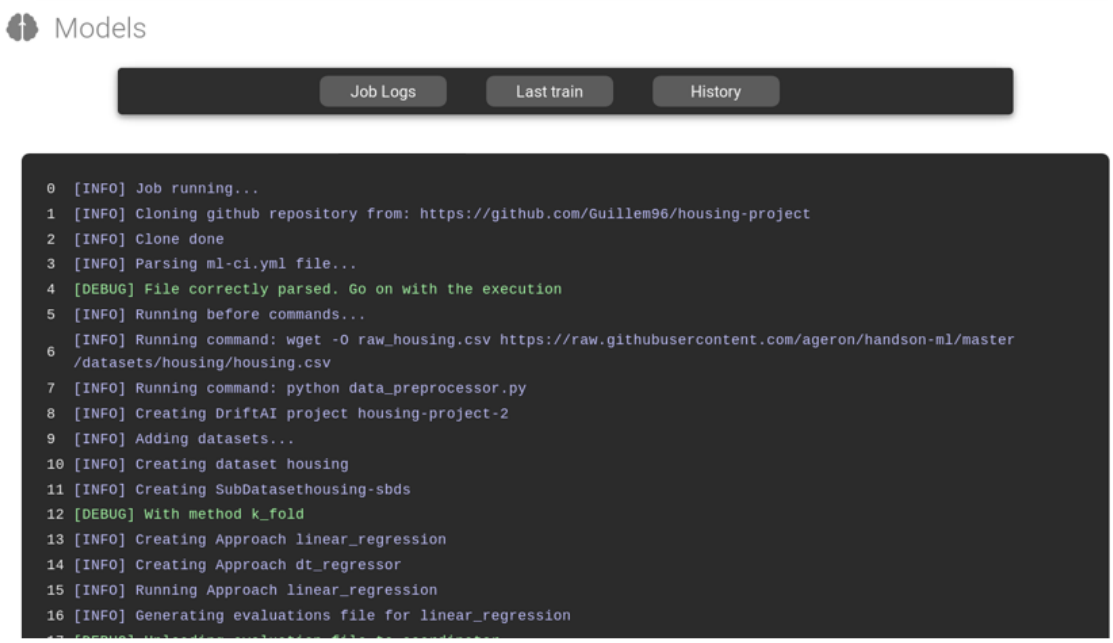

Figura 25: Registre logs corresponent a un entrenament en MlCi

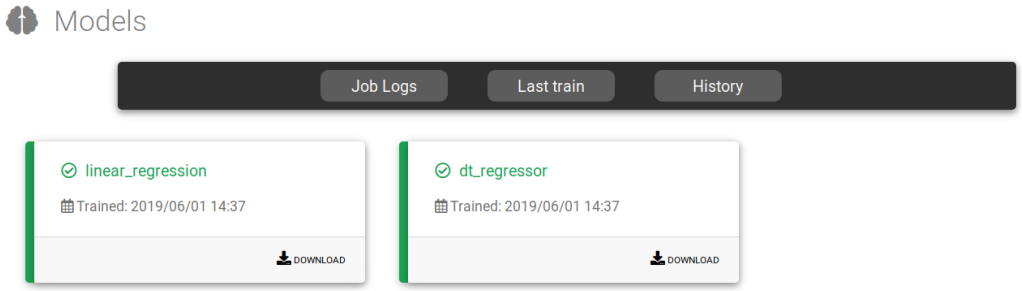

Figura 26: Estat execució RunnableApproaches en MlCi

El procediment per integrar MlCi només cal realitzar-lo un únic cop per a cada projecte. Un cop ja hem indicat a MlCi que mantingui el seguiment d'un dipòsit de GitHub cada vegada que es realitzi un commit, MlCi s'encarregarà d'executar el projecte de DriftAI definit en l'arxiu de configuració.

Si tot ha anat sota l'esperat, al cap d'aproximadament un minut, el projecte haurà canviat a l'estat *TRAINED* i tots els *approaches* tindran disponible el botó per realitzar la descàrrega del fitxer de les avaluacions finals per cadascuna de les combinacions de l'espai d'hyperparàmetres. Un cop premem el botó de descàrrega es mostra un pop up exemplificant una exploració dels resultats amb l'objectiu d'obtenir la millor configuració d' hyperparàmetres que maximitzen una mètrica determinada (Figura 27).

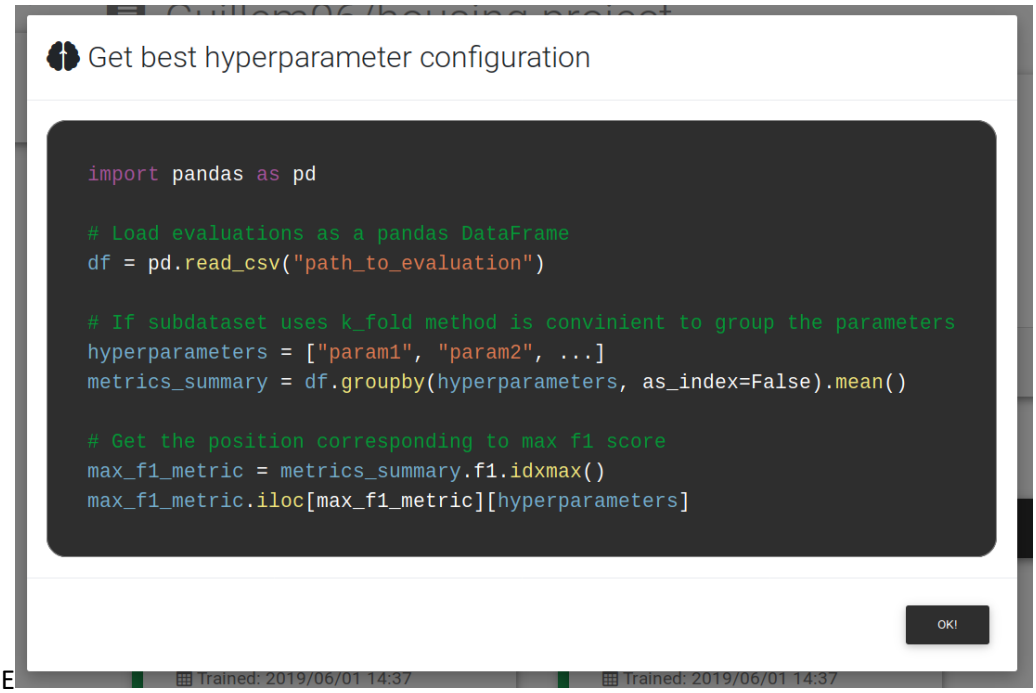

Figura 27: Descarrega evaluacions MlCi

Arquitectura MlCi

En aquest apartat s'explica l'arquitectura hardware de MlCi i les responsabilitats que té cada mòdul, juntament amb les tecnologies que han facilitat la seva implementació.

MlCi segueix l'arquitectura simple d'una aplicació client-servidor. El servidor està dividit en tres mòduls amb l'objectiu de separar les responsabilitats i, alhora, permetre que cada mòdul realitzi la seva tasca el més eficientment possible.

Per una banda, el client és una aplicació web desenvolupada amb Angular[34]. La seva funció tracta de mostrar els resultats dels entrenaments i també un històric de tots els *approaches*, juntament amb un enllaç per a la descàrrega de les seves avaluacions finals. A més, el client és l'encarregat d'enviar les credencials de *GitHub* al servidor una vegada l'usuari ha decidit iniciar sessió. S'ha utilitzat *Firebase*[35] per facilitar el procés d'autenticació.

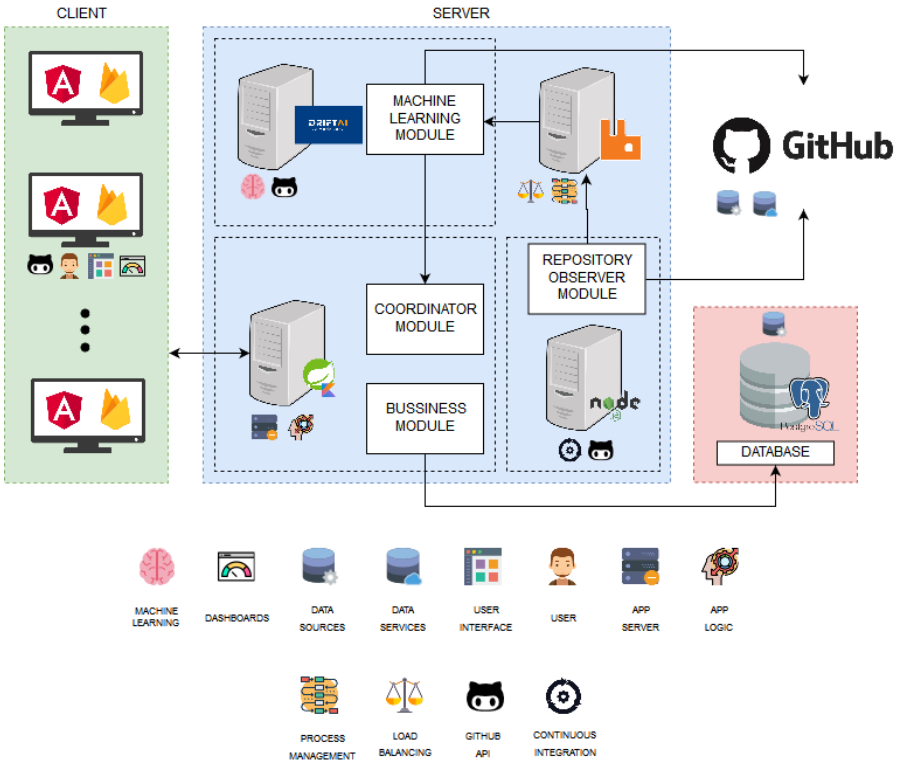

Figura 28: Arquitectura Hardware MlCi

Per altra banda, el servidor és la part complexa i és on es produeix el còmput necessari per assolir tots els processos d'entrenament i d'integració contínua. Com s'ha dit anteriorment, el servidor consta de tres mòduls principals:

[34] http://angular.io
[35] https://firebase.google.com/

- **Mòdul *Web Service:*** Aquest està dividit en dos submòduls. El primer s'anomena ***Coordinator*** i és l'encarregat de mantenir la comunicació entre els tres sistemes principals. El segon submòdul és el que conté la lògica del domini i té la finalitat de modelar el domini i mantenir la connexió amb la base de dades.
- **Mòdul de *Machine Learning:*** És el responsable de realitzar totes les tasques relacionades amb l'aprenentatge automàtic i, per tant, d'entrenar, avaluar i enviar els resultats a la unitat coordinadora per cadascun dels algorismes indicats en el fitxer de configuració. Per tal de facilitar l'execució del cicle de vida d'un projecte de ML, s'utilitza el mòdul DriftAI.

  El problema principal d'aquest sistema és que duu a terme tasques complexes i de llarga durada i, per aquest motiu, no és recomanable treballar amb el protocol HTTP. Així doncs, s'utilitza l'estàndard AMPQ i l'agent de missatges RabbitMQ[36] per enviar tasques a aquest servidor. En el cas del MlCi actual, existeixen dos productors (el mòdul *Repository Observer* i el client web) i un consumidor (la mateixa unitat de *Machine Learning (ML)*). Treballar amb un agent de missatges permet tenir una escalabilitat horitzontal molt senzilla. Per escalar el mòdul ML, simplement és necessari afegir més màquines o *workers* escoltant a la cua on es publiquen les tasques d'entrenament i, automàticament, es realitzarà el balanceig de tasques entre els diferents *workers.*
- **Mòdul *Repository Observer:*** L'objectiu d'aquest és observar contínuament el dipòsit de *GitHub* per a detectar canvis. Una vegada es detecta algun canvi, aquest notifica a la resta de sistemes per tal que s'iniciï una nova tasca d'entrenament.

[36] https://www.rabbitmq.com/

3.2.5. Contribucions a DriftAI

DriftAI es desenvolupa de forma àgil seguint una variació del *framework* SCRUM[37]. Les principals característiques que comparteix la metodologia de desenvolupament personalitzada per DriftAI i SCRUM són:

- **Tenen un product backlog**: Un product backlog és un llistat de tasques que s'han de dur a terme per tal de completar un projecte.
- Les tasques estan etiquetades per prioritat, de manera que les més prioritàries es realitzen abans que la resta.
- Diverses persones treballen en el mateix projecte. Això permet realitzar múltiples tasques en paral·lel.

El fet que DriftAI es desenvolupés amb aquesta metodologia va fer que m'hagués d'adaptar a l'equip i al codi ja existent. Tot i que en aquell moment DriftAI era una versió molt primitiva i poc estable, ja hi havia molt codi i algunes parts eren complexes d'entendre.

A continuació s'exposen les tasques i millores més rellevants que he aportat a DriftAI.

Documentació automàtica

Alguns dels problemes que hem vaig trobar al començar, va ser la manca de documentació en el codi, per tant una de les meves primeres aportacions va ser la preparació de l'entorn per tal generar la documentació de forma automàtica, així com anar documentant els diversos mètodes i classes que ja estaven implementats. Documentar el codi em va ajudar a entendre millor cadascun dels components que formaven part de DriftAI i també em va permetre millorar certs procediments.

Actualment, la documentació dels diferents mòduls de DriftAI es genera de forma automàtica gràcies a *Sphinx*[38]. L'estàndard que s'utilitza per redactar la documentació dels mètodes i les classes (*docstrings*[39]) s'anomena *Napoleon* (Fragment de Codi 1), estàndard també utilitzat per l'organització que manté *numpy*. A més, DriftAI disposa de documentació escrita manualment respecte a la seva instal·lació, al seu us i a components extra com ara el CLI.

Es pot consultar la documentació completa de DriftAI a través de l'enllaç referenciat a peu de pàgina[40].

[37] https://www.scrum.org/
[38] http://www.sphinx-doc.org/
[39] https://www.python.org/dev/peps/pep-0257/
[40] https://driftai.readthedocs.io/en/latest/

```
def  func ( arg1 ,  arg2 ):
    """Summary line.

    Extended description of function.

    Parameters
    ----------
    arg1 : int
        Description of arg1
    arg2 : str
        Description of arg2

    Returns
    -------
    bool
        Description of return value

    """
    return    True
```

Fragment de Codi 7: Napoleon docstring

Solució Orientada a Objectes

En les primeres versions DriftAI estava programat amb classes on cada classe tenia les seves responsabilitats. Això aportava certs beneficis de la programació orientada a objectes (OO). Tot i això, aquests possibles avantatges no s'estaven aprofitant al màxim i alguns principis SOLID com ara el Single *Responsibility* i el *Open/Closed, no* es complien fet que dificultaven les modificacions al codi.

En aquest punt, la meva aportació va ser reescriure parts del codi per tal de maximitzar els beneficis de la programació OO. Això va permetre adquirir a DriftAI una gran extensibilitat, el qual actualment és un dels seus principals forts.

En la Figura 29: Diagrama de classes simplificat es mostra el diagrama de classes simplificat resultant després de la meva aportació.

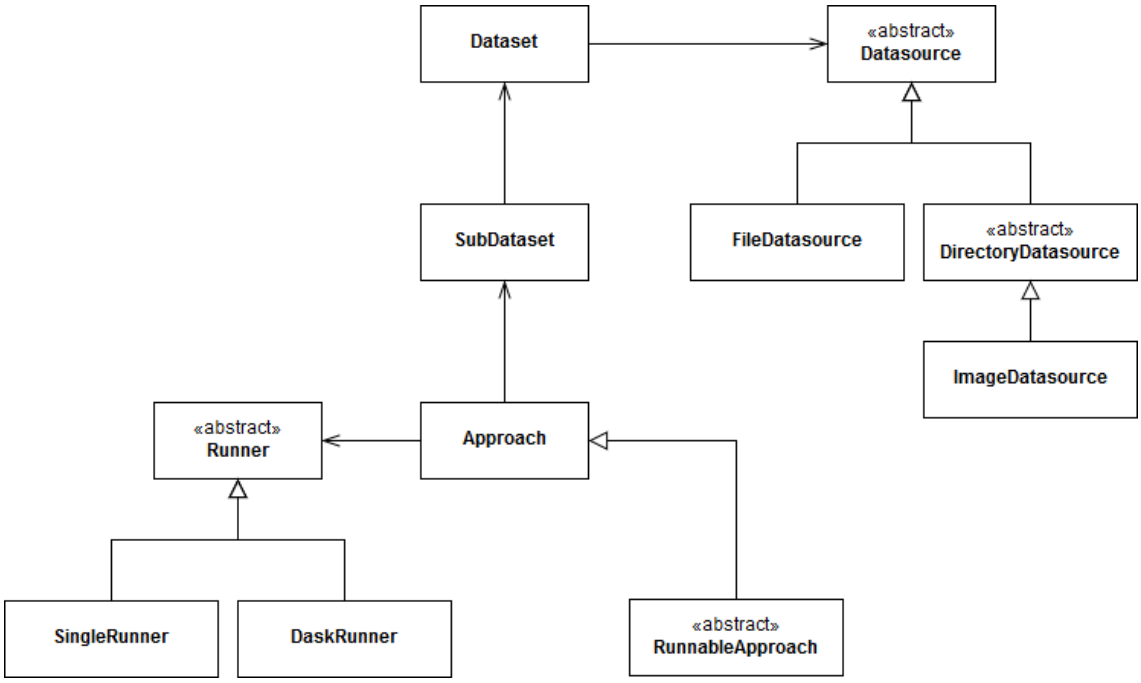

Figura 29: Diagrama de classes simplificat

En la Figura 29: Diagrama de classes simplificat es poden veure diverses classes que estan preparades per ser heretades per part dels usuaris finals i utilitzar-les en el seu benefici.

Per una banda tenim les classes abstractes *Datasource* o *DirectoryDatasource*, les quals heretar d'elles permet la carga dades de diferents orígens amb diferents formats. En l'exemple del *workflow,* internament DriftAI utilitza una instància de la classe *ImageDatasource* per carregar totes les imatges del MNIST, però en el cas que haguéssim d'entrenar un model utilitzant fitxers d'àudio, podríem estendre la classe *DirectoryDatasource* i sobreescriure el mètode *loader* de forma que cargues un fitxer d'àudio en un tensor o en un vector compatible amb el algorisme d'entrenament que posteriorment es decideixi utilitzar.

Per altra banda, hi ha la classe abstracta *Runner* que tal i com s'explica en l'apartat dels ingredients de DriftAI, un *runner* o *run engine* descriu com s'executa la cerca d'hyperparàmetres d'un *Approach*. Per defecte, DriftAI utilitza el *SingleRunner*, és a dir, que executa totes les *Runs* en local, però si es vulgues accelerar la cerca, es podria heretar de la classe *Runner* per tal de distribuir les diverses *Runs* en un clúster d'ordinadors núvol, per exemple, un clúster orquestrat per *kubernetes*[41].

[41] https://kubernetes.io/

Finalment, l'última classe abstracta s'anomena *RunnableApproach*, aquesta classe és la més important de tot DriftAI, ja que es la que utilitza l'usuari final per declarar les seves pròpies estratègies d'entrenament, predicció i espai d'hyperparàmetres a explorar. La explicació de com heretar de la classe *RunnableApproach* es pot trobar en l'apartat on s'explica el *workflow* de DriftAI.

Base de Dades Incrustada

Inicialment DriftAI no disposava d'una base de dades (BD) on emmagatzemar els índexs dels *Datasets*, les particions dels *Subdatasets,* les execucions i els seus estats, etc. Per emmagatzemar tota aquesta informació utilitzava fitxers formatats en l'estàndard JSON. Això comporta varies desavantatges:

- Lectures i escriptures lentes: Si partim de que un Approach pot tenir milers de *Runs* on cada *Run* era un fitxer, tant carregar com guardar modificacions comportava obrir i tancar milers de descriptors, accions que resulten ser molt lentes quan es realitzen a gran escala.
- Dificultat en mentir tants fitxers en un control de versions com ara *git*.
- Dificultat en referenciar els diferents objectes, ja que s'havia de fer mitjançant la seva ruta absoluta del sistema de fitxers. Per exemple l'identificador d'un *Dataset* podia ser *C:/Users/Guillem/Desktop/current-projects/ia/mnist-project/data/MNIST.*

El fet de migrar a una BD era més una necessitat que no una millora, ja que en *Aproaches* complexes, resultava inviable treballar amb una quantitat tant gran de fitxers.

Actualment DriftAI utilitza una BD incrustada anomenada TinyDB[42]. TinyDB és una BD incrustada NoSQL orientada a documents. Algunes de les avantatges comporta TinyDB són les següents:

- No té dependències externes (No depèn d'altres mòduls de python).
- API simple i intuïtiva
- Fàcilment extensible

Aprofitant el disseny OO descrit anteriorment, es defineix la classe abstracta *Peristent*, de la qual hereten tots els objectes de DriftAI. La classe *Persistent* descriu i implementa els següents mètodes:

- ***Save:*** Serialitza i emmagatzema un objecte.
- ***Load:*** Carrega un objecte de la BD i el parceja a un objecte de python.

[42] https://tinydb.readthedocs.io/en/latest/intro.html

- ***Update:*** Serialitza i actualitza un objecte ja existent a la base de dades.

A més DriftAI permet l'accés a les col·leccions de TinyDB mitjançant el mètode estàtic collection(). Per exemple, per explorar els diferents *Datasets* emmagatzemats en un projecte s'utilitza el Fragment de Codi 8.

```
from driftai .data import Dataset
all_datasets = Dataset.collection().all()
```

Fragment de Codi 8: Accés a una col·lecció de TinyDB

Command Line Interface (CLI)

En les primeres versions de DriftAI qualsevol acció s'havia de realitzar mitjançant scripts de python i hi havia certes funcionalitats que eren massa simples. Implementar aquestes funcionalitats en un script era trivial i molt mecànic.

Per millorar aquest aspecte, la meva solució va ser plantejar una CLI per interaccionar amb l'API de DriftAI, i d' aquesta forma substituir scripts simples però complexos de mantenir amb comandes que, transparentment, implementen la mateixa lògica, la qual serà actualitzada i mantinguda per l'equip desenvolupador de DriftAI.

Actualment, mitjançant el CLI es poden dur a terme pràcticament tots els passos necessaris per executar una cerca d'hyperparàmetres. Per exemple, amb 4 comandes simples el CLI de DriftAI es capaç de començar una cerca d'hyperparàmetres en un clúster d'ordinadors al núvol.

A continuació es fa un resum de totes les comandes de les quals disposa el CLI juntament amb la seva funcionalitat.

- ***dai new****:* Crea un projecte nou amb el nom especificat per paràmetre.

```
$ dai new my-dream-project
```

- ***dai add:*** Afegeix elements al projecte. Actualment es capaç d'importar dades de qualsevol *Datasource* i indexar-les en un *Dataset*.

```
$ dai add dataset -- path /path/to/my/data/ -- datatype img ...
```

- ***dai generate****:* Crea diversos elements d'un projecte. Actualment es capaç de generar *Subdatasets* i *Approaches.*

```
$ dai generate subdataset <dataset> -- method k_fold -- by 5
$ dai generate approach <approach_name> -- subdataset
<subdataset_id>
```

- ***dai run**:* Executa un *Approach* mostrant el seu progrés.

```
$ dai run <approach_name>
```

- ***dai status**:* Retorna el nombre de *Runs* pendents i executades.

```
$ dai status <approach_name>
```

- ***dai evaluate**:* Avalua els resultats d'un *Approach* i les exporta les avaluacions a un csv.

```
$ dai evaluate <approach_name>
```

Per conèixer més detalladament l'ús i possibilitats de les que disposa cada comanda es pot consultar la documentació[43] del CLI o executar la comanda:

```
$ dai <command>-- help
```

[43] https://driftai.readthedocs.io/en/latest/cli/cli.html

4. CONCLUSIONS

La intel·ligència artificial és un factor diferencial i un valor afegit per a les aplicacions d'avui en dia, i cada vegada més es tendeix a utilitzar-la amb la finalitat de millorar l'experiència de l'usuari final. Desafortunadament el procediment de creació de nous models és complex i no consisteix únicament a entrenar un algorisme, sinó que implica molts més procediments, especialment quan la IA és la columna vertebral d'un negoci. Per tant, és necessari establir un cicle de desenvolupament que garanteixi certa fiabilitat.

Quan parlem de desenvolupament software i aplicacions modernes apareixen característiques com les següents: escalabilitat, extensibilitat, modularitat, seguretat i bones pràctiques. Malauradament, en l'actualitat, en el cicle de vida de desenvolupament d'un model de IA aquestes característiques no s'apliquen, fet que comporta que una aplicació software que utilitzi un model de IA no pugui arribar a ser mai perfecta en termes d'escalabilitat, seguretat, etc., ja que el model no ho permet.

Per aquests motius, aquest projecte presenta DriftAI com a solució a tots els problemes esmentats. DriftAI permet que un model de IA escali juntament amb un negoci i l'aplicació *software* que es nodreix d'aquest.

En la Figura 30 podem observar el cicle de vida de desenvolupament d'un model de IA que contempla DriftAI.

Figura 30: Cicle de vida DriftAI

Tot i que la fase inicial queda fora del desenvolupament tècnic, DriftAI intenta automatitzar la resta de procediments mitjançant les bones pràctiques basades en experiències anteriors.

Els punts forts del *pipeline* de creació d'un nou model en DriftAI són els següents:

- Abstracció de l'origen de dades gràcies a la representació abstracta d'un *Datasource*, és a dir, utilitzant DriftAI podem treballar amb les dades sense saber d'on i com les estem extraient. Això permetrà reutilitzar tot el codi de tractament de dades i entrenament encara que la font de dades canvi.
- Accés a les dades estandarditzat. Per accedir a les dades s'utilitzen els seus índexs, és a dir, s'utilitza un identificador únic per referenciar a una instància del *dataset.* Així

facilitem el control de la memòria i el tractament de les dades tant a l'hora d'entrenament com d'avaluació.

- Separació entre conjunts de validació i entrenament estandarditzats. Gràcies a l'accés indexat s'estableixen mètodes de separació que divideixen les dades en múltiples conjunts per tal d'evitar el filtratge d'informació entre els grups d'entrenament i validació.
- Entrenament juntament amb la cerca de la millor configuració d'hyperparàmetres. DriftAI permet entrenar un algorisme i a la vegada buscar la millor configuració d'hyperparàmetres, tot de forma transparent a l'usuari final.
- Avaluació dels models estandarditzada. Totes les mètriques que proporciona DriftAI disposen de la mateixa interfície, fet que facilita la creació de mètriques personalitzades.
- Preparat per créixer en el futur. Un dels punts forts més importants de DriftAI és la seva extensibilitat. Això permet a tot desenvolupador adaptar el cicle de vida a les seves necessitats.

5. TREBALL FUTUR

En aquest apartat s'expliquen totes les millores que estan previstes per a DriftAI en un futur pròxim.

Dotar a DriftAI de la capacitat de resoldre problemes més tipus de problemes:

- Resoldre problemes de *weak supervision*, que tot i que es pot entendre com un cas particular de problemes supervisats, es podrien resoldre una afegint un pas intermig de generació de *weak labels.*
- Resoldre problemes no supervisats, juntament amb la problemàtica que comporta avaluar els resultats obtinguts en algorismes de *clustering* o de *density estimation.*

Aspectes a millorar del preprocessat de dades:

- Permetre la creació de *pipelines* de preprocessament de dades i *feature engineering*.
- Incorporar a DriftAI més *Datasources,* per exemple, un que tingui accés als *buckets de Google Cloud*[44].

Millores aplicables a l'entrenament i a la cerca d'hyperparàmetres:

- Incorporar a DriftAI més *Run Engines,* especialment un que permeti la distribució de les tasques en un clúster d'ordinadors.
- Generar un espai d'hyperparàmetres de forma automàtica.
- Implementar la cerca d'hyperparàmetres amb diferents estratègies, com ara optimització bayesiana. Actualment en DriftAI només està disponible la cerca mitjançant *grid search CV.*

Afegir mètodes per posar en producció un model:

- Dotar a DriftAI d'un desplegament de models intel·ligent, és a dir, tenint en compte el model que està actualment desplegat, desplegar el nou model sempre que sigui millor que l'anterior.
- Dissenyar i implementar un mòdul que permeti el seguiment del model en producció, per tal de detectar alguna anomalia en el seu comportament o un decreixement del seu rendiment.

[44] https://cloud.google.com/storage/

6. REFERENCIES BIBLIOGRÀFIQUES

Bergstra, James, Dan Yamins, and DD Cox. 2013. “Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms.” *PROCEEDINGS OF THE 12th PYTHON IN SCIENCE CONFERENCE* (Scipy).

C. Martin Robert. 2000. *Design Principles and Design Patterns*. https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf (May 5, 2019).

Chollet, Francois. 2017. *Deep Learning with Python*. Publications, Manning.

Fawcett, Tom. 2006. “An Introduction to ROC Analysis.” *Pattern Recognition Letters*.

Geron, Aurelien. 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. ed. O’Reilly Media.

Goodfellow, Ian et al. 2014. “Generative Adversarial Nets.” *Corrosion*.

Hinton, Geoffrey E, David E Rumelhart, and Ronald J Williams. 1986. “Learning Representations by Back-Propagating Errors.” *Nature*.

de Jonge, Edwin, and Mark van der Loo. 2013. “An Introduction to Data Cleaning with R.” *Statistics Netherlands, The Hauge*.

Jouppi, Norman P et al. “In - Datacenter Performance Analysis of a Tensor Processing Unit NETWORKS.”

Landset, Sara, Taghi M. Khoshgoftaar, Aaron N. Richter, and Tawfiq Hasanin. 2015. “A Survey of Open Source Tools for Machine Learning with Big Data in the Hadoop Ecosystem.” *Journal of Big Data* 2(1): 24.

Lecun, Yann, Le’on Bottou, Yoshua Bengio, and Parick Haffner. 1998. “Gradient-Based Learning Applied to Document Recognition.” *proc. OF THE IEEE*.

Moore, By Gordon E. 1975. “Cramming More Components onto Integrated Circuits.”

Pedregosa, Fabian et al. 2011. “Scikit-Learn: Machine Learning in Python.” *Journal of Machine Learning Research* 12(Oct): 2825–30.

Robertson, Suzanne, and James Robertson. 2012. *Mastering the Requirements Process: Getting Requirements Right*. 3rd ed. ed. Addison Wesley.

Stackoverflow. 2019. “Stack Overflow Developer Survey 2019.”

https://insights.stackoverflow.com/survey/2019 (May 30, 2019).

Van Der Walt, Stefan, S. Chris Colbert, and Gaël Varoquaux. 2011. “The NumPy Array: A Structure for Efficient Numerical Computation.”